Music Metadata Capture in the Studio from Audio and Symbolic Data

Thesis submitted in partial fulfilment
of the requirements of the University of London
for the Degree of Doctor of Philosophy

Steven Hargreaves

September 2014

School of Electronic Engineering and Computer Science,
Queen Mary, University of London

I, Steven Hargreaves, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third partys copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Signature:

Date: 28th September 2014

Details of collaborations and publications:

Journal Article

S. Hargreaves, A. Klapuri, and M. Sandler. Structural segmentation of multitrack audio. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(10):2637 –2647, dec. 2012. ISSN 1558-7916

Conference Paper

Steven Hargreaves, Geraint Wiggins, and Mark Sandler. A semantic web approach to pattern discovery in data and music. In *Audio Engineering Society Conference: 53rd International Conference: Semantic Audio*. Audio Engineering Society, 2014

The following research papers, published during the timeframe of this PhD research, are not included in this thesis:

Conference Paper

Steven Hargreaves, Chris Landone, Mark Sandler, and Panos Kudumakis. Segmentation and discovery of podcast content. In *Audio Engineering Society Conference 128*. Audio Engineering Society, 2010

Post Symposium Proceedings (as co-author)

Mathieu Barthet, Steven Hargreaves, and Mark Sandler. Speech/music discrimination in audio podcast using structural segmentation and timbre recognition. In Sølvi Ystad, Mitsuko Aramaki, Richard Kronland-Martinet, and Kristoffer Jensen, editors, *Exploring Music Contents*, volume 6684 of *Lecture Notes in Computer Science*, pages 138–162. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23125-4

Abstract

Music Information Retrieval (MIR) tasks, in the main, are concerned with the accurate generation of one of a number of different types of music metadata – beat onsets, or melody extraction, for example. Almost always, they operate on fully mixed digital audio recordings. Commonly, this means that a large amount of signal processing effort is directed towards the isolation, and then identification, of certain highly relevant aspects of the audio mix. In some cases, results of one MIR algorithm are useful, if not essential, to the operation of another – a chord detection algorithm for example, is highly dependent upon accurate pitch detection. Although not clearly defined in all cases, certain rules exist which we may take from music theory in order to assist the task – the particular note intervals which make up a specific chord, for example.

On the question of generating accurate, low level music metadata (e.g. chromatic pitch and score onset time), a potentially huge advantage lies in the use of multitrack, rather than mixed, audio recordings, in which the separate instrument recordings may be analysed in isolation.

Additionally, in MIR, as in many other research areas currently, there is an increasing push towards the use of the Semantic Web for publishing metadata using the Resource Description Framework (RDF). Semantic Web technologies, though, also facilitate the querying of data via the SPARQL query language, as well as logical inferencing via the careful creation and use of web ontology language (OWL) ontologies. This, in turn, opens up the intriguing possibility of deferring our decision regarding which particular type of MIR query to ask of our low-level music metadata until some point later down the line, long after all the heavy signal processing has been carried out.

In this thesis, we describe an over-arching vision for an alternative

MIR paradigm, built around the principles of early, studio-based metadata capture, and exploitation of open, machine-readable Semantic Web data. Using the specific example of structural segmentation, we demonstrate that by analysing multitrack rather than mixed audio, we are able to achieve a significant and quantifiable increase in the accuracy of our segmentation algorithm. We also provide details of a new multitrack audio dataset with structural segmentation annotations, created as part of this research, and available for public use.

Furthermore, we show that it is possible to fully implement a pair of pattern discovery algorithms (the SIA and SIATEC algorithms – highly applicable, but not restricted to, symbolic music data analysis) using only Semantic Web technologies – the SPARQL query language, acting on RDF data, in tandem with a small OWL ontology. We describe the challenges encountered by taking this approach, the particular solution we've arrived at, and we evaluate the implementation both in terms of its execution time, and also within the wider context of our vision for a new MIR paradigm.

In memory of Ian Hargreaves

Acknowledgements

Firstly, thanks go to my supervisors – Mark Sandler, Anssi Klapuri, and Geraint Wiggins, for invaluable guidance, support and advice over the duration of my research studies. Thanks also to everyone in the Centre for Digital Music, and in particular, to Dan Stowell, Holger Kirchoff, and Ken O'Hanlon for feedback on early thesis drafts, and to Matthias Mauch, Sebastian Ewert, György Fazekas, Mathieu Barthet, Luis Figueíra, Magdalena Chudy, Alice Clifford, Peter Foster, Jordan Smith, Robert Tubb, Thomas Wilmering, Katy Noland, Emmanouil Benetos, Chunyang Song, Christopher Harte, Wen Xue, Yading Song and Chris Cannam for a variety of technical, practical and social services.

For having the foresight to buy me a computer, in the days before that was commonplace, for affording me the opportunity to explore music, and for too many other things to mention – thanks to my parents. Last but not least, thanks to my flatmate Alison for her long-term moral support. This work has been supported by EPSRC studentship no. EP/505054/1.

License

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Much commercially recorded music follows a typical path from performance to release. Although there are exceptions, such as live recordings of jazz and classical music, the path is frequently:

1. Instruments are recorded separately, and digitally.
2. Recordings are mixed by an engineer or producer.
3. The stereo mix is released.

At some later date, researchers, musicologists or industry might attempt, computationally, to extract various types of semantic metadata from the released version of the recording – this is commonly referred to as Music Information Retrieval (MIR). Whether attempting to automatically transcribe the audio, find the onsets of percussive beats, perform instrument classification or any other MIR task, a significant amount of theoretical and computational effort will be devoted to the isolation of certain musical phenomena of interest from the complete ensemble mix. Given the prevalence of studios employing digital recording techniques, a question arises – why not perform MIR tasks earlier in the production chain, prior to mixing, when we still have access to individual instrument recordings?

Depending on the degree to which we are able to increase the accuracy of MIR algorithms by using multitrack data, this should, in principle, result in a symbolic, or close to symbolic, representation of recorded music. Furthermore, as we shall see in Sections 2.3.4 and 2.3.5 of Chapter 2, there

is an increasing trend, both generally and within the MIR community, towards utilising the Semantic Web as a means of publishing metadata in a machine-readable format. The technologies that constitute the Semantic Web though are not solely concerned with metadata publishing – they also provide mechanisms for both querying, and making new inferences from, existing data.

We hypothesise that a rich set of symbolic music metadata expressed using the Resource Description Framework (RDF), and containing a similar level of detail to that commonly found in Musical Instrument Digital Interface (MIDI) data, would be of great value. Additionally, in combination with the use of carefully engineered ontologies and/or SPARQL queries (see Section 2.3), such a set of metadata would facilitate valuable further musicological insights into the original recording later down the line. As new questions arise concerning the content of a piece of music, instead of embarking upon further, potentially complex, audio analysis, we may instead formulate new queries for our RDF symbolic music metadata. Consequently, after setting out our vision of a new MIR paradigm based upon early, accurate, signal processing-based metadata generation in the studio, and the subsequent querying of lightweight symbolic Semantic Web metadata, this thesis explores two main themes:

1. The question of whether or not there is a quantifiable advantage to be gained by performing an MIR task using multitrack, rather than mixed, audio.

2. The viability of deriving new, perceptually significant insights from symbolic music data, using Semantic Web technologies alone.

When researching the first of these two themes, there are a number of potential MIR tasks we could choose. In many cases, when the metadata we seek to extract is an attribute of a distinct subset of the total set of recorded instruments (such as automatic transcription or beat tracking), the case for using multitrack audio seems fairly clear. In the case of one common MIR task though, that of structural segmentation, the benefits are not quite as clear cut. Imagine attempting to locate the chorus and

verse segments of a pop song from the bass or keyboard recordings alone – depending on the particular song in question, this could either be trivial or impossible. Consequently, we choose structural segmentation as our test MIR task, and in Chapter 4 we present the results of our multitrack structural segmentation experiment. An additional outcome of this part of our research is a new publicly available dataset[1], containing structural segmentation annotations of 104 multitrack audio recordings.

Following on from this, rather than conducting additional multitrack audio MIR tasks in a similar vein, we take something of a leap of faith. Moving to the symbolic data domain, under the assumption that at some point in the future the ability to derive a sufficiently rich and accurate symbolic representation of recorded audio will be feasible, we implement and evaluate the performance of a pair of pattern discovery algorithms, SIA and SIATEC (Meredith et al., 2002), using Semantic Web technologies. These algorithms are particularly pertinent in that they are inherently applicable to multidimensional data, and, therefore, given that channel number or instrument type could be mapped to one particular dimension, whilst other attributes such as score time and chromatic pitch could be mapped to others, they are ideally suited to the analysis of multichannel symbolic data.

1.1 Motivation

Our primary motivation for conducting this research is the wealth of seemingly fertile ground to be exploited where multitrack audio is concerned. So much popular music is recorded digitally, and so much effort is made within the MIR community to 'reverse engineer' the work done during the mixing phase, that we believe it is imperative to at least begin re-focusing the target of MIR algorithms. This argument, as we shall see in Section 2.1 of Chapter 2, is only strengthened by the apparent 'glass ceiling' currently being witnessed when attempting to increase the accuracy of many MIR algorithms.

[1] `http://c4dm.eecs.qmul.ac.uk/rdr/handle/123456789/36`

Secondly, we feel strongly that it is important to conduct research which evaluates the computational capabilities of Semantic Web technologies, given the current level of interest in their adoption. Although many authors have described methods for sharing linked metadata and the use of Semantic Web ontologies within MIR (we provide details in Section 2.3.5), to the author's knowledge, no attempts have been made to actually perform algorithmic analysis of symbolic music data using only the Semantic Web technologies themselves. An exception is the use of a 'Harmony' ontology (Ibbotson, 2009) from which one may infer the temporal precedence of given combinations of chords and/or keys. The SIA and SIATEC algorithms belong to a commonly encountered class of 3SUM-hard (Clifford et al., 2006), cross-product type algorithms (other examples are given in Chapter 5), and furthermore, no empirical data exists regarding the computational complexity of SPARQL 1.1 (which, as we shall see in Chapter 5, forms the backbone of our implementation) when applied to this type of algorithm.

1.2 Scope

This thesis is not an attempt to describe a fully comprehensive, end-to-end, audio to symbolic data representation of music. Rather, it is an exploration of the gains to be made by utilising multitrack rather than mixed audio when performing structural segmentation, as well as an investigation into the viability of applying the Semantic Web technologies RDF, SPARQL 1.1, and OWL 2, to the task of deriving new, perceptually relevant information from a symbolic representation of music data. During our structural segmentation experiments, we limit ourselves to rock and pop music.

1.3 Specific Contributions

- Empirical evidence that structural segmentation accuracy may be significantly improved by using multitrack rather than mixed audio recordings. This result was published in the IEEE Transactions on Audio, Speech and Language Processing journal (Hargreaves et al., 2012).
- A human-annotated structural segmentation ground-truth dataset of multitrack audio, containing 104 songs[2], publicly accessible for re-use by other researchers.
- Proof-of-concept evidence that a pattern discovery algorithm involving complex, compound data structures, can successfully be fully implemented using only Semantic Web technologies, together with performance evaluation metrics (Hargreaves et al., 2014, in print) and full implementation details.

1.4 Thesis Structure

The rest of this thesis is set out as follows:

Chapter 2 – Background

This chapter provides the core background material upon which the main body of research is based. We show evidence of a reduction in accuracy for some important MIR tasks when they are applied to mixed, rather than single, instrument recordings, and we present a brief overview of some of the techniques commonly used to locate the structural segments of music. We introduce pattern discovery in symbolic music data, and provide a general overview of the Semantic Web.

[2] `http://c4dm.eecs.qmul.ac.uk/rdr/handle/123456789/36`

Chapter 3 – A Vision of a New MIR Paradigm

Expanding on the material presented in the Background chapter, we present an over-arching vision for an alternative MIR paradigm, built around the principles of early, studio-based metadata capture, and exploitation of open, machine-readable Semantic Web data.

Chapter 4 – Structural Segmentation of Multitrack Audio

In this chapter we describe, and present the results of, two experiments designed to evaluate the effect of using multitrack, rather than mixed, audio when attempting to locate the structural segment boundaries of rock and pop music recordings. We also present details of a new structural segmentation ground-truth dataset of multitrack audio.

Chapter 5 – A Semantic Web Approach to Pattern Discovery in Data and Music

In Chapter 5 we explore the viability of using only Semantic Web technologies in order to derive new, perceptually relevant data from multidimensional symbolic music score data. We describe the particular method used, evaluate its performance when compared to a more conventional approach, and discuss the difficulties and challenges involved.

Chapter 6 – Conclusions

Finally in Chapter 6 we present our overall conclusions, suggestions for further work, and describe some potential applications of the results of this thesis.

Chapter 2

Background

In this chapter we make the case for performing certain MIR tasks earlier in the production chain; that is, whilst we still have access to the individual instrument, multitrack recordings. We provide evidence that some MIR tasks produce less accurate results from mixed multi-instrument audio recordings than from single-instrument recordings, and observe that accuracy levels over recent years from techniques based upon mixed audio are not increasing significantly. Later, in Chapter 4, we will demonstrate improved structural segmentation accuracy via the use of multitrack audio; consequently we provide some more in-depth background here on the subject of structural segmentation. Based then upon the assumption that multitrack audio-based MIR brings us closer to the possibility of being able to produce accurate symbolic representations of multi-channel music, we also describe some methods for performing further analysis of symbolic music data, both single and multi-channel. Additionally, we discuss the growing trend towards the use of RDF data as a means of sharing metadata. Together, these two themes of symbolic music data analysis and RDF metadata form the basis for Chapter 5, in which we demonstrate how we may perform multi-channel symbolic data pattern discovery using purely Semantic Web technologies.

2.1 Single versus Multi-Instrument Music Information Retrieval

Pitch detection for monophonic sources, often referred to as single F0 estimation, is a well-established and still very active area of research, particularly within the speech analysis domain. In a real music audio signal though, it is far more common that multiple sources will be present simultaneously. The problem of multiple F0 estimation is much harder, and has received comparatively less attention. As an example, one approach, taken by Klapuri (2004), involves repeatedly cancelling out each detected pitch from a signal until we are satisfied that we have detected all F0s. Clearly the execution time alone of such an algorithm will be worse than one which assumes only a single F0 is present.

A useful illustration of the relative accuracies of single and multiple pitch detection algorithms can be found by comparing the results of two closely related and recent pitch-detection algorithms by the same authors. Benetos and Dixon (2011) use shift-invariant probabilistic latent component analysis, constrained by a Hidden Markov Model (HMM) to detect pitches in monophonic music excerpts. They extend the model in Benetos and Dixon (2012) with multiple HMMs providing temporal constraints, and multiple-instrument spectral templates. Although the test datasets are necessarily different, it is reasonably clear from the results shown in Tables 2.1 and 2.2 that both accuracy and error rates are significantly worse in the multi-pitch case.

Method	**Instrument**	Acc	E_{tot}	E_{subs}	E_{fn}	E_{fp}
Left to Right HMM	Piano	81.5%	17.8%	2.2%	9.8%	5.8%
	Cello	80.3%	22.1%	8.3%	5.6%	15.7%
	Oboe	55.0%	39.1%	13.3%	22.6%	3.2%

Table 2.1: Single-pitch detection results from (Benetos and Dixon, 2011) for three different single instruments

Despite the fact that the results for solo oboe are relatively poor, in the cases of solo piano and cello, accuracy is at least 20% higher and the various error percentage metrics are significantly lower compared to

Dataset	Acc_1	E_{tot}	E_{subs}	E_{fn}	E_{fp}
RWC	61.6%	37.2%	9.1%	18.3%	9.8%
Disklavier	58.6%	42.7%	9.9%	16.3%	16.5%
MIREX	41.0%	53.0%	25.4%	20.1%	7.5%

Table 2.2: Multi-pitch detection results from (Benetos and Dixon, 2012) for three different polyphonic datasets – RWC (Goto et al., 2003), Disklavier (Poliner and Ellis, 2007), and the woodwind quintet recording from the MIREX multi-F0 development set

the multi-pitch detection algorithm. The definitions of the metrics used (which come from Poliner and Ellis, 2007) follow. Overall accuracy is defined as:

$$ACC = \frac{TP}{(FP + FN + TP)} \tag{2.1}$$

where TP ("true positives") is the number of correctly transcribed voiced frames (over all notes), FP ("false positives") is the number of unvoiced note-frames transcribed as voiced, and FN ("false negatives") is the number of voiced note-frames transcribed as unvoiced. This measure is bounded by 0 and 1, with 1 corresponding to perfect transcription. There are four different types of error measure, in each of which the intersection of N_{sys} reported pitches and N_{ref} ground-truth pitches counts as the number of correct pitches N_{corr}. The total error score, integrated across all time frames t, is:

$$E_{tot} = \frac{\sum_{t=1}^{T} max(N_{ref}(t), N_{sys}(t)) - N_{corr}(t)}{\sum_{t=1}^{T} N_{ref}(t)} \tag{2.2}$$

Substitution error is defined as:

$$E_{subs} = \frac{\sum_{t=1}^{T} min(N_{ref}(t), N_{sys}(t)) - N_{corr}(t)}{\sum_{t=1}^{T} N_{ref}(t)} \tag{2.3}$$

The "false negative" error is:

$$E_{fn} = \frac{\sum_{t=1}^{T} max(0, N_{ref}(t) - N_{sys}(t))}{\sum_{t=1}^{T} N_{ref}(t)} \tag{2.4}$$

and the "false positive" error is:

$$E_{fp} = \frac{\sum_{t=1}^{T} max(0, N_{sys}(t) - N_{ref}(t))}{\sum_{t=1}^{T} N_{ref}(t)} \tag{2.5}$$

This is just one example; in order to see the more general picture it is instructive to examine the results of the annual Music Information Retrieval Evaluation eXchange (MIREX[1]) evaluation tasks. Figure 2.1 shows the trajectory of multiple F0 accuracy (as defined in 2.1) from 2007 to 2012 – after an initial rise between 2007 and 2008, there follows a noticeable plateau, and even a slight degradation in 2012.

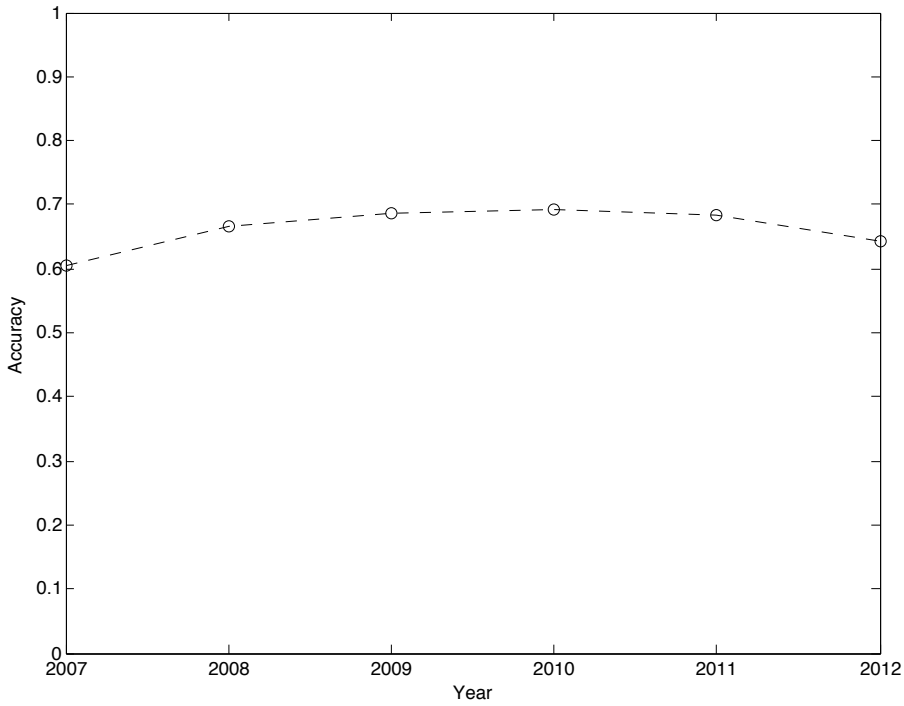

Figure 2.1: Historical MIREX multiple F0 detection accuracy results

Similar trends are evident in the chord-detection (Figure 2.2 – unfortunately no easily locatable definition of the metric used here is to be found on the MIREX website) and structural segmentation (Figure 2.3 – see Equation 4.6 in Section 4.4.2 for the definition of boundary retrieval f-measure) tasks. The evaluation dataset used for the chord-detection task has changed over the years – before 2009 it was only the Beatles dataset provided by Harte et al. (2005). From 2009, 38 more songs by Queen and

[1] `http://www.music-ir.org/mirex`

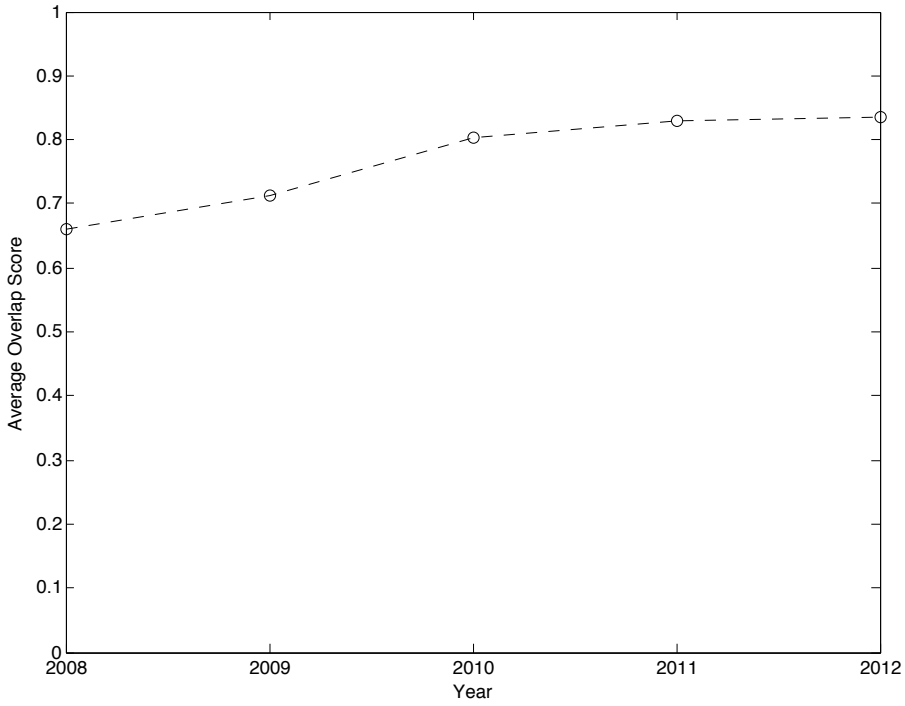

Figure 2.2: Historical MIREX chord detection average overlap score results

Zweieck were added, and approximately 200 songs from Burgoyne et al. (2011) were incorporated in 2012. It is perhaps reasonable to speculate that perceived algorithmic advances are being kept in check by more representative datasets.

Another vibrant area of MIR research is audio source separation – the difficult task of 'un-mixing' a mixed audio recording into its constituent instruments (sources), or sometimes the perhaps slightly less daunting task of separation into harmonic and percussive components. Ono et al. (2008), motivated by their assertion that percussive tones interfere with multi-pitch analysis, whilst suppression of harmonic components aids rhythm analysis, present a real-time algorithm which separates the harmonic and percussive components of an audio signal – the Harmonic-Percussion Signal Separation (HPSS) algorithm. In a similar vein, but using a different method, Fitzgerald (2010) tries to achieve the same kind of separation, proposing that this will be a useful pre-processing stage for "automatic

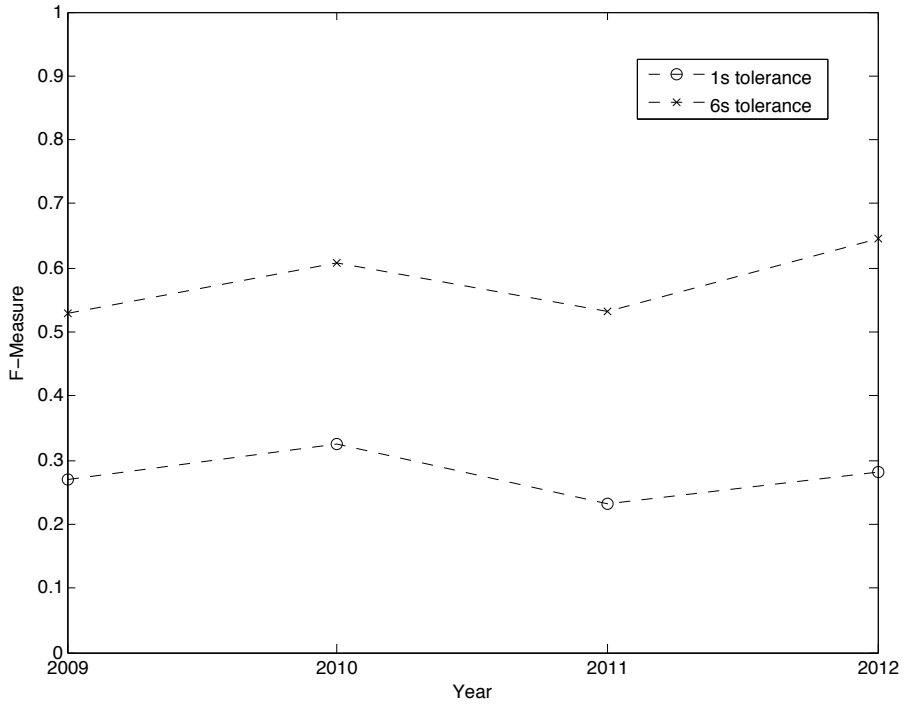

Figure 2.3: Historical MIREX structural segmentation boundary retrieval f-measure results for 1s and 6s tolerances

transcription of pitched instruments, key signature detection and chord detection". Ueda et al. (2010) employ an HMM-enhanced version of (Ono et al., 2008) to perform automatic chord detection, achieving the highest rank in the Audio Chord Detection task of MIREX 2008,[2] whilst Rump et al. (2010) also use the HPSS algorithm, this time in order to achieve improved Mel Frequency Cepstral Coefficient (MFCC) based genre classification accuracy. Tsunoo et al. (2010) extend the technique from (Ono et al., 2008) in order to extract the percussive and bass components of audio signals with a view to achieving improved mood classification. As impressive as some of the results in this area are, the question remains – given that in many cases it is possible to access the multitrack sources, is this a sensible direction of effort?

Some authors (Pachet and Aucouturier, 2004; Downie, 2008; Benetos et al., 2012) in the MIR community speculate upon the existence of a 'glass

[2] http://www.music-ir.org/mirex/2008

ceiling', an apparent limit to the accuracy we might realistically hope to achieve using common algorithmic techniques and evaluation datasets. It is with this in mind that we apply our focus in this thesis to the exploitation of alternative source data types rather than the pursuit of small percentage accuracy gains via modest algorithmic parameter and/or method adjustments.

2.2 Structural Segmentation of Audio

The preceding argument motivates us in Chapter 4 to demonstrate a significant improvement in the accuracy of a structural segmentation algorithm utilising multitrack audio. In preparation for that, in this section we provide some segmentation background material and discussion of related works.

Structural segmentation of audio is the task of locating the temporal locations of the boundaries between the perceptually distinct, medium to long time-frame sections of a piece of music (i.e. in the order of at least one musical bar, although in some cases we may be interested in sub-bar-level segments too). For example, in western rock and pop music, it is common to refer to the chorus, verse and bridge segment of a song. All occurrences of segments to which one would apply the same label (e.g. verse) are regarded as perceptually similar, whilst segments having different labels are not.

An immediate difficulty inherent in this task is that we only have a fuzzy definition of what constitutes these high-level segments. For example, whilst the notion of identifying the verses in a pop song is something that many people are familiar with, specifying a precise description of what a verse is, is not easy. If two identical chord progressions, lasting eight bars, occur twice within a song, we would probably apply the same label to them. However, if a pitch-transposed, or 12 bar long, version of the chord progression occurs somewhere else, we would probably apply to the same label to that too. More subtle differences, such as changes in instrumentation, or the presence or absence of individual cymbal crashes, would probably not cause a human listener to perceive these segments as

significantly different to each other. Depending upon the method of analysis undertaken though, a segmentation algorithm may well regard these segments as being unrelated. At best, we can say that at least one perceptually significant aspect of two similarly perceived segments must be shared between the two – be that the underlying chord progression, timbre, melody, or rhythm. Any attempt to identify the structural segments of music is usually evaluated on the basis of comparing machine generated segment boundaries with those identified by one or more human listeners (the 'ground truth').

Despite these difficulties, several researchers have nevertheless described methods of carrying out structural segmentation of music. Abdallah et al. (2005) employ an unsupervised Bayesian clustering model to classify signal frames according to their audio properties. In their case, audio properties are obtained by calculating a constant-Q log-power spectrum, the dimensionality of which is then reduced using principal component analysis. Aucouturier et al. (2005) use MFCCs as the audio feature, and a Gaussian mixture model to estimate the distribution of these features, Mauch et al. (2009) search for repetition of chroma sequences, whilst Barrington et al. (2010) describe a dynamic texture model based upon both timbral and rhythmical features. We concentrate here on the common themes of audio features, self-distance (alternatively known as self-similarity) matrices, homogeneity detection, and repetition detection. The reader is referred to Paulus et al. (2010) for a comprehensive overview of music structure analysis techniques.

2.2.1 Audio Features

In order to start making meaningful inferences about the music represented by an audio signal, it is common to first transform it into quantifiable measures which are more closely aligned with human perception of music than simple amplitude variations (although amplitude does play an important role in music perception). This is the process of converting the audio signal into a sequence of audio feature vectors $\boldsymbol{v_1}, \boldsymbol{v_2}, ... \boldsymbol{v_T}$, and there are numerous types of audio features which may be of interest to us.

Root Mean Square Energy

Perhaps the easiest feature to extract is root mean square (RMS) energy. Tzanetakis and Cook (1999) identify the relevance of the RMS energy audio feature to segmentation by noting that it is related to loudness, and changes in loudness are important cues for new sound events. The RMS energy of one audio frame is given by

$$e(i) = \sqrt{\frac{\sum_{t=1}^{t=n} y(t)^2}{n}} \tag{2.6}$$

where i is the audio frame index, $y(t)$ is the amplitude of the signal at sample t within the frame i, and n is the number of discrete time sampled signal amplitude values in frame i. Unlike the following audio features, RMS energy is in fact a scalar. For the purposes of comparison to other audio features, we treat it as a one-element vector.

Chroma

The chroma representation of pitch, proposed by Shepard (1964) indicates the relative levels of each of the 12 notes of the equal-tempered chromatic scale present in an audio sample, without indicating the octave to which each note belongs (an alternative explanation is also given by Bartsch and Wakefield, 2005). Clearly this has direct relevance to analysis of western music; the ability to determine the relative strengths of each note as time varies offers us the potential to identify both melody and harmony as well as the repetition and variation of sequences of notes, phrases and chord progressions. Chroma audio features have been successfully utilised in applications such as chorus identification (Bartsch and Wakefield, 2005; Goto, 2006), music thumbnailing (Bartsch and Wakefield, 2005; Chai and Vercoe, 2003), and cover song identification (Ellis and Poliner, 2007; Ravuri and Ellis, 2010). For the single, i^{th}, frame of audio, the twelve elements of the chroma feature vector are given by

$$c_k(i) = \sum_{f \in S_k} \frac{X_i(f)}{N_k}, \quad k \in \{1...12\} \tag{2.7}$$

where $X_i(f)$ is the logarithmic magnitude of the discrete Fourier transform (DFT) of the audio frame, every $S_k \in \mathbb{Z}$ defines, for every pitch class k, a

subset of the discrete frequency space, and N_k is the number of elements belonging to S_k.

Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCCs) are audio features which are commonly used to quantify the "timbre" of a sample of audio. Timbre itself is not clearly defined, but is typically taken to be an indication of the "quality" of a sound. The American National Standards Institute provide this definition: "Timbre is that attribute of auditory sensation in terms of which a listener can judge that two sounds similarly presented and having the same loudness and pitch are dissimilar" (ANSI, 1960). In the context of solo instruments we use the term to describe the unique sound of a particular instrument (for example the sound of a clarinet compared to that of a saxophone), and beyond that, we would also talk about the difference in timbre of different instruments of the same class, in order to distinguish (for example) one clarinet from another, or the playing styles of different musicians. In a more general sense we use the term 'polyphonic timbre' to describe the overall sound or texture of mixed, polyphonic audio (Aucouturier et al., 2005); for example we might say that each bar of a verse in a pop song has similar timbre, whilst the timbres of the verse and the chorus differ. From a technical point of view, MFCCs are calculated by, firstly, determining the log-power in a series frequency bands. These bands are chosen to relate closely to the critical bands of the human ear. As an aid to achieving this, the centre frequencies of these bands are picked from the Mel, rather than the linear, frequency scale. The linear frequency scale may be mapped to the Mel scale as follows:

$$mel = 2595 \log_{10} \left(1 + \frac{f}{700}\right) \tag{2.8}$$

where f is the frequency in Hertz and mel is the Mel frequency. The MFCC feature vector coefficients are then calculated by discrete cosine transforming the log-power spectrum

$$m_k(i) = \sum_{b=0}^{B-1} E_i(b) \cos \frac{\pi(2b+1)k}{2B} \tag{2.9}$$

where i is the frame index, k is the coefficient index, b is the band number, B is the total number of bands, and $E_i(b)$ is the log energy of band b for frame i. Timbre modeling is the key feature used by Aucouturier et al. (2005) to perform segmentation via long-term similarity and pattern identification.

Rhythmogram

Jensen et al. (2005) describe an audio feature, the rhythmogram, which quantifies the degree of rhythmic change within a piece of audio. First, we calculate the Perceptual Spectral Flux as

$$\delta(i) = \sum_{k=1}^{N/2} W(f_k)\{(a_k^i)^{1/3} - (a_k^{i-1})^{1/3}\} \tag{2.10}$$

where i is the frame index, a_k and f_k are the magnitude and frequency of the bin k of the short-time Fourier transform (STFT) obtained using a Hanning window, and N is the STFT length. W is the frequency weighting used to represent an equal loudness contour. The rhythmogram itself is then calculated using autocorrelation over a short time window (e.g. 2 seconds) from

$$r_k(i) = \sum_{j=i}^{i+l} \delta(j)\delta(j+k) \tag{2.11}$$

where l is the length of the summing window, i is the frame index, and k is the feature vector coefficient index for frame i.

Other features which may be of interest include those suggested by Tzanetakis and Cook (1999) for multi-feature segmentation (namely spectral centroid, spectral roll-off, spectral flux, and zero crossings), and normalised constant Q spectra subjected to Principal Component Analysis (used by Levy et al. 2006 and Abdallah et al. 2005 for high-level music structure analysis).

Careful selection of either a single type of audio feature, or, as suggested by Tzanetakis and Cook (1999), a combination of features, allows us to proceed to a study of the higher levels of music information contained within the audio signal; for example the variations and repetitions of pitch, melody, dynamics, chords, harmony and so forth. Segment boundaries

themselves are often indicated by significant changes of multiple features (Bregman, 1994). More evidence of the usefulness of multiple features is given by Bruderer (2008), who finds that important cues are harmonic progressions, change in timbre, change in tempo and change in rhythm.

2.2.2 Self-Distance Matrices

Audio features alone do not present us with a structural segmentation of musical audio – we must perform further processing of these features in order to deduce the locations of regions of similarity or repetition. One possible step in this process is to employ a widely used technique known as self-distance (or alternatively self-similarity) matrix calculation, as proposed by Foote (2000). Using a suitable distance measure such as the cosine angle between two audio feature vectors, we define the self-distance between frame i and frame j as

$$D(i,j) = 0.5\left(1 - \frac{\boldsymbol{v}_i \cdot \boldsymbol{v}_j}{||\boldsymbol{v}_i||\ ||\boldsymbol{v}_j||}\right) \tag{2.12}$$

where $\boldsymbol{v}_i$ is the feature vector associated with frame i, and i and j are frame indexes, the signal is compared with itself in terms of one or more audio features.

The result of calculating these distance measures across all feature vectors is a two-dimensional matrix. By assigning different colours to the values in this matrix we are able to produce an informative visualisation of the self-similarity in the audio signal. An example derived from the chroma features of the song "People let's stop the war" by Brad Stanfield (a pop/rock song with clear chorus, verse and bridge sections) is shown in Figure 2.4. The temporal locations of the ground truth segment boundaries are shown above the self-distance matrix; a clear correlation can be seen between the ground truth locations and the vertical lines dividing regions of homogeneous colour in the matrix image.

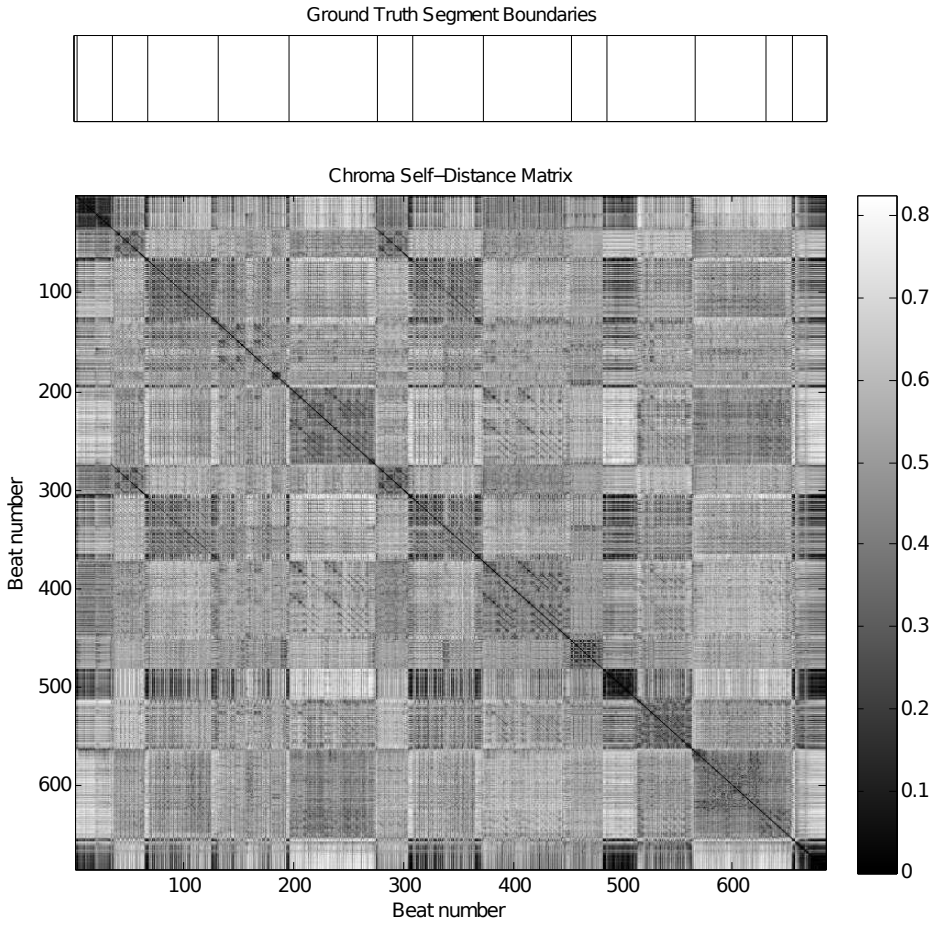

Figure 2.4: Ground truth segments (top) and visualisation of a self distance matrix for a musical audio signal (bottom)

2.2.3 Beat-Aligned Frames

When we later come to pick out segment boundaries from this self-distance matrix, the temporal accuracy at which we are able to operate will inevitably be limited by the length of the audio frames we use to calculate each audio feature vector. As long as these frames correspond to sufficiently short periods of time we will be able to pinpoint temporal locations to a desirable level of accuracy. Intuitively we might expect that boundaries are more likely to fall on strong beats (as opposed to either weak or no beats), and research into boundary perception by Bruderer (2008) supports this hypothesis. Consequently it would be helpful if we

were to choose the length of the audio frames such that they correspond to beat intervals present in the audio. By first analysing the audio using a beat tracking algorithm, we are then able to choose lengths such that any frame we select as a boundary is guaranteed to coincide with a beat (assuming the results of the beat analysis are sufficiently accurate). This enables us to cope with changes in tempo by varying the frame lengths in accordance with variations in the distances in time between beats. Furthermore, the number of elements present in self-distance matrix, and also therefore execution time, is significantly reduced.

2.2.4 Homogeneity Detection

Visualisations of self-distance matrices offer a valuable insight into the structure of a piece of music. We still need, however, to perform further analysis of the data in order to derive a set of segment boundaries. Noting that areas of homogeneity are represented as square or rectangular blocks in the visualisations, Foote (2000) proposes a method wherein we determine the variation in the level of correlation (the 'novelty score') between a simple binary checkerboard pattern (a kernel) and the self-distance matrix as we slide the kernel along the main diagonal of the self-distance matrix. Explicitly, we first create an $n \times n$ kernel matrix C (the two-by-two case is shown in Equation 2.13).

$$C = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.13}$$

The time scale upon which variations can be detected is proportional to the size of the kernel, and so if we require larger kernels, they are formed by taking the Kronecker product of C and a matrix of ones, e.g. (again, using the two-by-two example)

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \bigotimes \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \tag{2.14}$$

The novelty score $s(i)$ is then calculated as:

$$s(i) = \sum_{m=-L/2+1}^{L/2} \sum_{n=-L/2+1}^{L/2} C(m,n)D(i+m,i+n) \tag{2.15}$$

where i is the frame number, L is the width (lag) of the kernel C centered on (0,0) and D is the self-distance matrix. Peaks in the novelty score correspond to significant position changes in our multidimensional feature space. Consequently, locating segment boundaries becomes a matter of determining which of the peaks represent a sufficiently large change in feature space position as to constitute a segment boundary. The novelty score derived from the same chroma features used for Figure 2.4 is shown in Figure 2.5, along with the same ground truth segment data. Again, good, although not perfect, agreement between the ground truth segments and the peaks in the novelty score can be seen. Segment boundaries are therefore found by employing some method of selecting the peaks in the novelty score; in our case we try two different methods for comparison purposes (see Section 4.4.1).

2.2.5 Repetition Detection

As an alternative to searching self-distance matrices for regions of homogeneity, we may also look for repeat sequences, which manifest themselves as stripes (diagonal lines off the main diagonal). This is the technique employed by Mauch et al. (2009) in the algorithm used as a benchmark later in this thesis. After constructing a self-distance matrix from beat-synchronous chroma features, candidate segments are identified by searching for stripes. Computation time is reduced by assuming a minimum segment length of 12 beats, and a maximum of 128. Only beats exhibiting a correlation above an empirically derived threshold value are considered as segment beginnings, and further refinement is achieved via the calculation of "likely bar beginnings"; local maxima in the convolution of a function representing likelihood of harmonic change with a kernel of spikes every 2 beats. Finally a greedy algorithm is used to decide which of the candidate

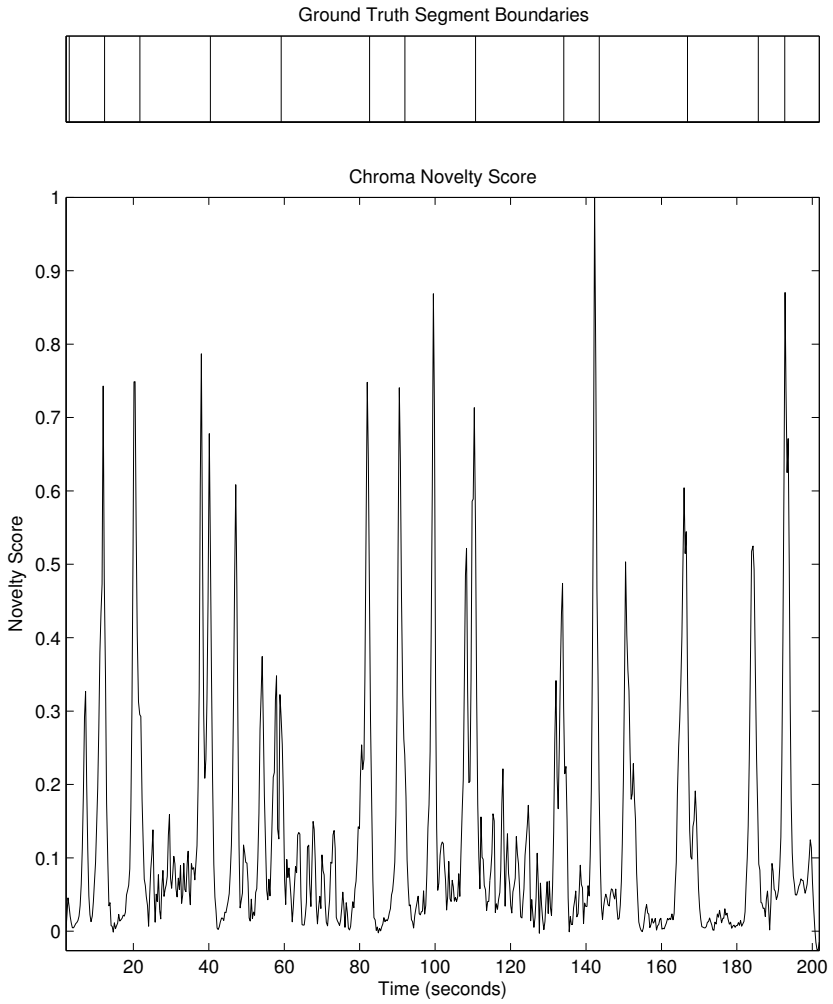

Figure 2.5: Ground truth segments (top) and normalised novelty score for a musical audio signal (bottom)

segments are true segments.

2.2.6 Hidden Markov Models

Hidden Markov Models (HMMs) enable us to determine the probability that a system is in a particular state q_t at time t, given that we have observed some other variable x_t, know the transition probabilities from q_{t-1} to q_t, and know the emission probability for x_t given q_t. They have been successfully applied to pattern recognition applications such as speech

recognition (Rabiner 1989 provides a good tutorial and a comprehensive list of further references in this field), whilst some authors have also used them as an alternative (or in some cases, such as Peeters et al. 2002, in addition) to using self-distance matrices, to reveal the structure of music. Aucouturier and Sandler (2001) show that using HMMs can be beneficial when attempting to segment complex music such as classical, however in the simpler case of rock/pop their use is unnecessary. Raphael (1999) also uses HMMs to segment classical music, but at the lower level of individual notes and rests. The scope of the structural segmentation experiments described in Chapter 4 is limited to rock and pop music, and so we choose not to employ HMMs in our segmentation technique.

2.3 The Semantic Web

So far we have presented some evidence of the limitations of using mixed audio when conducting certain MIR experiments (Section 2.1), as well as some of the common methods used in order to structurally segment audio (Section 2.2). This serves as a precursor to then utilising multitrack audio in Chapter 4 in order to obtain one aspect (structural segments) of a more accurate, over-arching symbolic representation of recorded music. We may then ask: assuming we are able to extract a more 'complete' symbolic representation of music from multitrack audio (e.g. one with a comparable level of detail to MIDI), is it possible to not only perform further analysis of this symbolic data, but to share both the results and our algorithmic methods on the web too, along with (importantly) the *semantics* of the data, according to a commonly agreed specification? Doing so would allow other agents, be they client applications or other web services, to consume, analyse, and contribute to our data without recourse to any specific Application Programming Interface (API), vendor implementation or machine architecture. Consequently, in the following sections we provide, firstly, an overview of what we mean by 'The Semantic Web', and what its capabilities are (Sections 2.3.1, 2.3.2, and 2.3.3), followed by some examples of scientific and engineering disciplines currently using Semantic Web technologies (Sections 2.3.4 and 2.3.5), which forms our motivation

for applying Semantic Web technologies to the task of pattern discovery in symbolic music data (Chapter 5).

In this thesis, when we refer to 'Semantic Web Technologies', we are referring collectively to the Resource Description Framework (RDF), the SPARQL query language ("SPARQL" is a recursive acronym for SPARQL Protocol and RDF Query Language), and the Web Ontology Language (OWL). These technologies are beginning to mature (the original RDF specification was published in 2004), and as such, alongside the specifications[3,4,5], a wealth of tutorial material exists (Segaran et al., 2009; Passin, 2004; Davies et al., 2006; Leuf, 2006; Antoniou, 2004; Allemang and Hendler, 2011). Particularly useful are the online RDF and OWL 'primers'[6,7]. Nevertheless, in the interests of completeness and readability, a brief overview of the theory and workings of Semantic Web technologies follows.

2.3.1 Resource Description Framework (RDF) Data

The Resource Description Framework (RDF) is a language in which we may represent knowledge about resources in the World Wide Web in the form of a collection of *triples* . There are three components to each triple: the *subject*, *predicate*, and *object*. The subject refers to the *thing* which we are describing, the predicate is some property of the subject, and the object identifies the value of the predicate. Importantly, the object of one triple may also be the subject of another, allowing us to *link* data.

[3] `http://www.w3.org/standards/techs/rdf`
[4] `http://www.w3.org/TR/sparql11-query/`
[5] `http://www.w3.org/TR/owl2-syntax/`
[6] `http://www.w3.org/TR/rdf-primer/`
[7] `http://www.w3.org/TR/owl2-primer/`

Given that we are representing knowledge about resources on the web, each of the three components of a triple takes the form of a Uniform Resource Identifier (URI). As an example, we could represent the assertion that something on the web has the name "John Smith" using the following three URIs, where the first represents the subject of the triple, the second the predicate, and the third the object:

Subject: `http://www.person.com/id#abc123`

Predicate: `http://www.hr.com/name`

Object: `"John Smith"^^<http://www.w3.org/2001/XMLSchema#string>`

The subject and predicate here are standard URIs. In many cases the object may also be a standard URI, however, in this case, because our object is some raw data, we use a general URI which represents the raw string datatype (the `^^<http://www.w3.org/2001/XMLSchema#string>` part of the URI) in combination with some characters enclosed by quotes. An RDF parser will interpret this syntax as representing the raw string "John Smith" – a *typed literal* [8].

Although not strictly necessary, it is common practice and extremely useful to use the same subsection of a URI to group conceptually equivalent resources. For example, if our triple above were part of a human resources database, we would also want to store the names of many other people, and it makes sense to use a common URI stub or (formally) *namespace* to which we append a unique suffix for each person. We would also want to store more than just the name of each person. By defining the three namespaces shown in Table 2.3, we may then use a shortened notation to represent our triple:

Subject: `person:abc123`

Predicate: `hr:name`

Object: `"John Smith"^^xsd:string`

[8] `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#typedliterals`

Namespace Prefix	Namespace URI
person	`http://www.person.com/id#`
hr	`http://www.hr.com/`
xsd	`http://www.w3.org/2001/XMLSchema#`

Table 2.3: Some example namespaces and their prefixes

Further triples may then be added easily using these same namespaces, for example, another person, this time "Mary Jones":

Subject: `person:def321`

Predicate: `hr:name`

Object: `"Mary Jones"^^xsd:string`

It might be the case that Mary Jones is John Smith's line manager, and we could represent this fact with another triple:

Subject: `person:def321`

Predicate: `hr:manages`

Object: `person:abc123`

Note that in this triple the object is a standard URI rather than a typed literal. Moreover, the object of this triple (our web resource representing the person whose name is "John Smith") is also the subject of our first triple. This is what is mean by *linked data* – we are now starting to build a *directed graph* of data. In an RDF graph, both subjects and objects are considered to be *nodes*, whilst predicates are directed arcs connecting subject nodes to object nodes. We have also made use of a new predicate, 'manages', belonging to the same namespace as our 'name' predicate, which we use to assert that the *thing* represented by a given subject 'manages' the *thing* represented by the given object. Figure 2.6 illustrates this more clearly.

In contrast to a relational database, in which the types of data we may store and the relationships between the data are dictated by a database schema, this model allows complete flexibility regarding which 'facts' we may assert, even to the point of permitting us to assert contradictory information. The motivation behind this approach is driven by a recognition that information represented by existing resources on the world wide web, such as HyperText Markup Language

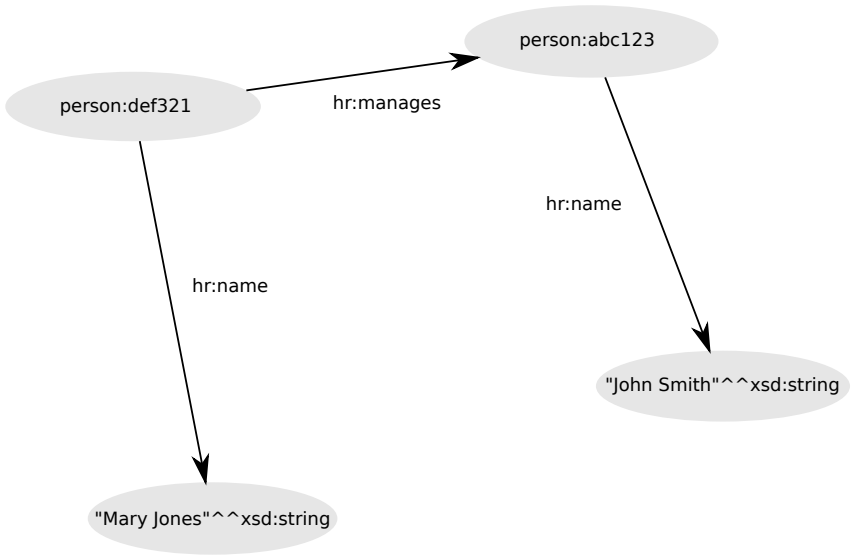

Figure 2.6: A simple RDF (linked data) directed graph

(HTML) documents, cannot be guaranteed to be accurate or non-contradictory, but nevertheless do clearly contain a wealth of useful information. Equally, no database schema can ever possibly be declared to be perfectly designed and future-proof. Instead, in the RDF model, it is accepted that anything can be said about anything, and extensive use is made of ontologies (see Section 2.3.3) in order that we may make sense of at least some subset of the asserted facts available to us.

2.3.2 The SPARQL Query Language

SPARQL, much like the Structured Query Language (SQL) for relational databases, allows us to query a collection of triples (a *triple store*). We may select variables from one or more graphs, where certain conditions (which we must specify) hold. The language allows us to conditionally filter out certain results, to perform aggregation over variables, to perform the union of two or more sets of query results, and to bind the results of arithmetic operations to variables. As an example, acting upon the small collection of triples we have so far (Figure 2.6), if we wished to find out which 'things' are managed by other 'things', we

could run the following query:

```
PREFIX hr: <http://www.hr.com/>

SELECT ?employee
WHERE
{
    ?manager hr:manages ?employee
}
```

The first line of this query declares the `hr` namespace we wish to use, whilst elsewhere, tokens preceded by `?` are variables. The `where` clause of the query specifies that we are searching the triple store for any triples in which the predicate is `hr:manages`. The presence of variables (`?manager` and `?employee`) in the subject and object positions indicate that we don't mind what values appear there. Once we have a list of triples matching these conditions, we select all values of `?employee` from that list (i.e. all the object parts of the matched list of triples). In our case, this yields a single result (shown here in its fully expanded form):

```
http://www.person.com/id#abc123
```

We might also wish to know the names of these managed people – this can be achieved by adding another condition to the 'where' clause and selecting a different variable:

```
PREFIX hr: <http://www.hr.com/>

SELECT ?name
WHERE
{
    ?manager hr:manages ?employee .
    ?employee hr:name ?name .
}
```

We are now selecting all the objects `?name` such that some subject `?manager` has a predicate `hr:manages` and an object `?employee`, and that same `?employee` appears as the subject in at least one other triple, which itself has a predicate

`hr:name` and object `?name`. This time the result is a single object:

```
"John Smith"^^<http://www.w3.org/2001/XMLSchema#string>
```

Finally, if we wished to know the names of all the 'things' in our triple store, regardless of whether they are managers or managed, we could relax the constraints in our query a little:

```
PREFIX hr: <http://www.hr.com/>

SELECT ?name
WHERE
{
    ?person hr:name ?name .
}
```

We are now simply searching for all the object parts of any triples in which the predicate is `hr:name`, which yields two results:

```
"Mary Jones"^^<http://www.w3.org/2001/XMLSchema#string>
"John Smith"^^<http://www.w3.org/2001/XMLSchema#string>
```

These are some very simple SPARQL query examples – as we mentioned at the start of this section there are many other more complex operations we may perform. The reader is referred to the SPARQL specification[9] or any of the many Semantic Web tutorial textbooks available (e.g. Segaran et al., 2009; Allemang and Hendler, 2011) for a more comprehensive overview.

2.3.3 The Web Ontology Language (OWL)

Whereas RDF allows us to represent simple facts as triples, OWL goes further and allows us to express the *meaning* of information. It enables this by providing mechanisms by which we may express relationships between facts such as class membership, class equivalence, property domain and property range. The OWL relationships themselves are also expressed as RDF, and it is left to an OWL implementation (or *reasoner*) to process the knowledge represented in a triple store (basic RDF facts as well as OWL relationships) according to the OWL semantics and rules. Depending on the level of OWL conformity for a particular implementation, this will include a certain level of *inference* capability; inferring new facts from the combination of basic RDF statements and

[9] `http://www.w3.org/TR/sparql11-query/`

the relationships we describe using OWL RDF statements. As with RDF and SPARQL, we do not present a comprehensive description of OWL here, instead referring the reader to the specification[10], primer[11], and tutorial textbooks (e.g. Segaran et al., 2009; Passin, 2004; Davies et al., 2006; Leuf, 2006; Antoniou, 2004; Allemang and Hendler, 2011). It is instructive however to demonstrate some of the main concepts with worked examples.

So far we've been able to assert facts in the form of triples, link data (triples), and perform queries on our set of triples. Without making some assumptions based upon any recognisable english words used in our RDF data though, we cannot draw any conclusions about the intended meaning of our data. For example, we have two separate subjects which both have the 'hr:name' predicate, but does that mean these two subjects, conceptually speaking, are the same kind of 'thing'? At present there is no way to tell, and a machine parsing our RDF wouldn't even be able to make any kind of english language-based assumptions. OWL, in combination with RDF Schema[12] (RDFS), allows us to make the relationships between our triples explicit. As an example, in our human resources triple store, we might have a mixture of permanent employees and contractors, and we might want to know which 'things' can be classed as 'Contractor Managers' – i.e. something which manages a contractor. First we'll add some more triples to our triple store:

```
person:abc456 hr:name "David Thompson"^^xsd:string .
person:def456 hr:name "Helen Rogers"^^xsd:string .

person:abc456 hr:manages person:def456 .
person:abc123 rdf:type hr:Contractor .
```

We now have have two more subjects, with names "David Thompson" and "Helen Rogers". We've also asserted that one of these subjects `manages` the other, and, that one of our original subjects, `person:abc123` (who has the `hr:name` "John Smith"), has a predicate `rdf:type` and corresponding object `hr:Contractor`. A human reader might be able to deduce from all of these triples that we have two managers now, but only one of them (Mary Jones) manages any contractors. Indeed, we could write a SPARQL query which would

[10] `http://www.w3.org/TR/owl2-syntax/`
[11] `http://www.w3.org/TR/owl2-primer/`
[12] `http://www.w3.org/TR/rdf-schema/`

appear to answer the question of 'which managers manage contractors':

```
PREFIX hr: <http://www.hr.com/>

SELECT ?name
WHERE
{
    ?person hr:name ?name .
    ?person hr:manages ?employee .
    ?employee rdf:type hr:Contractor
}
```

The result of this query is "Mary Jones", as we would hope. The reality at this point though is that we've only defined the specification of a contractor manager in our SPARQL query – there's nothing in the RDF data itself which would allow a machine to infer that Mary Jones belongs to a *Class* of "Contractor Managers". Amongst other things, OWL allows us to define *Restrictions*, which we may use to define class membership according to certain restrictions on the values of predicates. Let us add the following triples to our triple store, where owl represents the namespace `http://www.w3.org/2002/07/owl#`:

```
hr:ContractorManager owl:equivalentClass
  [ rdf:type           owl:Restriction;
    owl:onProperty     hr:manages;
    owl:someValuesFrom hr:Contractor] .
```

This is a more complex expression of triples than those we've used so far – we have a subject `hr:ContractorManager`, a predicate `owl:equivalentClass`, and then an object, the definition of which is contained within square brackets and spread across the next three lines. The first part of the contents of the square brackets is a space character, which denotes a blank node, or *bnode*. A bnode can be used when we don't actually need to refer to a permanent URI – we just need to allocate a dynamically generated node for use within the current scope of our semantic definitions or queries. We then assert three triples, all of which have this bnode as their common subject. Taken as a whole, the expression above states that if any triples exist which have a predicate `hr:manages`, and at least one corresponding object which itself has an `rdf:type` of `hr:Contractor`, then we may infer that the subject of that triple belongs to

the class `hr:ContractorManager`. Now we may perform a different query:

```
PREFIX hr: <http://www.hr.com/>

SELECT ?name
WHERE
{
    ?person rdf:type hr:ContractorManager .
    ?person hr:name ?name
}
```

The result is the same ("Mary Jones") – significantly though, this time the fact that Mary Jones belongs to the class `hr:ContractorManager` (and also that another manager, David Thompson, does not) has been *inferred* from the collection of triples in our triple store, some of which are simple assertions, and some of which define more complex semantics using OWL. Again, we refer the reader to the OWL specification[13], primer[14], or textbooks (Segaran et al., 2009; Passin, 2004; Davies et al., 2006; Leuf, 2006; Antoniou, 2004; Allemang and Hendler, 2011) for a more comprehensive overview of the full capabilities of OWL.

2.3.4 The Proliferation of the Semantic Web

In the preceding sections we described what the Semantic Web is, and gave an overview of its capabilities. In this section we present some real-world examples of its use.

The British Broadcasting Corporation (BBC) is increasingly making use of Semantic Web technologies in order to produce a larger amount of news and media-related web content from smaller levels of journalistic input[15]. By making extensive use of ontologies for concept categorisation, together with appropriately annotated media content, web pages centred around a particular topic may be generated with little or no input from a journalist. For example, an event such as the London Olympics involves a large number of athletes from all around the world, many of whom are not well known outside of their own country. Additionally, large amounts of performance results and news items will be generated as the games progress. Detailed coverage of all individuals in

[13] `http://www.w3.org/TR/owl2-syntax/`
[14] `http://www.w3.org/TR/owl2-primer/`
[15] `http://www.bbc.co.uk/academy/technology/software-engineering/semantic-web`

all competitions would require journalistic resources beyond the BBC's usual capacity, but with well designed ontologies and careful use of linked metadata, web pages composed of multiple, related items concerning (for example) one specific athlete, may be either completely auto-generated or at the very least presented to a journalist in a semi-complete state for rapid editing and approval.

In the field of bioinformatics, scientists employed by different companies often work on similar research projects, using different equipment and techniques, across different parts of the world. Experiments, for example in gene expression and microarray data, are continually leading to new insights regarding the biological function of genes. The need to use and maintain controlled vocabularies in this context is crucial but also hard to guarantee. Elements of OWL such as the `owl:sameAs` predicate facilitate the use of synonyms when searching multiple datasets. The ongoing production of high volume data also means that the accepted relationships between genes and their biological functions is continually in a state of flux. Consequently a number of publicly available ontologies are under constant development, for example the National Cancer Institute Thesaurus[16] and the gene ontology[17] (strictly speaking, the full gene ontology is not a Semantic Web ontology, although a filtered version is available in the OWL format).

In the interests of transparency, the United Kingdom government (along with governments of other countries) is making much of its non-personal, non-sensitive data publicly available[18], much of which can be downloaded in RDF format. Consequently the potential exists for otherwise disparate datasets, such as coastal bathing water quality from the environment agency and road traffic flow rates from the department for transport, to be queried as one integrated dataset, possibly leading to new insights into phenomena such as the causes and effects of population behaviour, or geographical variations in health.

2.3.5 The Proliferation of Semantic Audio

Closer to the research area of this thesis, the recently published "Roadmap for Music Information Research" (Serra et al., 2013) lists "Extend the scope of

[16] `http://ncit.nci.nih.gov/`
[17] `http://www.geneontology.org/`
[18] `http://data.gov.uk/`

existing ontologies" as one of the specific challenges of music representation. Several ontologies have already been produced, and some researchers advocate the use of the Semantic Web in applications such as artist metadata (Raimond et al., 2007), studio production workflow (Fazekas and Sandler, 2011; Fazekas, 2012), audio effects control and use (Wilmering et al., 2011), chord annotation[19], and audio feature annotation[20].

The first of these ontologies to appear was the music ontology (Raimond et al., 2007). The name is too broad – music, fundamentally, is a human, psychological phenomenon, in which we make cognitive perceptions in response to certain types of auditory signals (Wiggins et al., 2010). The music ontology does not deal with these phenomena at all – rather, it is concerned with the cataloguing of complete musical works, be they recorded or live performances, or symbolic scores. Figure 2.7 shows the terms used in the music ontology.

The audio features ontology[20] facilitates annotation of lower-level (i.e. smaller time-frame) features of an audio signal. Building upon the timeline[21] and event ontologies[22] (amongst others), it allows us to represent the characteristics of an audio signal within a particular time interval, such as chromagram features, pitch, onsets, speech segments and amplitude.

As the name suggests, the chord ontology[23] provides terms for representation of the notes, and intervals between notes, that go to make up particular chords. The question of whether or not any particular ontology sufficiently and accurately reflects the domain it is intended to represent is a difficult one to answer – all of these ontologies have been constructed manually by authors who believe they will be of value to other potential users within the intended domain. As a side note, in an effort to reduce or remove human error or bias during ontology design, some authors describe methods of automatic or semi-automatic ontology generation (Kolozali et al., 2011; Kolozali, 2013; Jordanous, 2010).

Leaving aside the question of utility for the moment, taken in combination, these ontologies provide us with the means to create extremely rich sets

[19] `http://purl.org/ontology/chord/`
[20] `http://purl.org/ontology/af/`
[21] `http://purl.org/NET/c4dm/timeline.owl#`
[22] `http://purl.org/NET/c4dm/event.owl#`
[23] `http://purl.org/ontology/chord/`

Music Ontology At A Glance

An alphabetical index of Music Ontology terms, by class (categories or types), by property and by individuals. All the terms are hyperlinked to their detailed description for quick reference.

Classes: | AnalogSignal | Arrangement | AudioFile | CD | Composition | CorporateBody | DAT | DCC | DVDA | DigitalSignal | ED2K | Festival | Genre | Instrument | Instrumentation | Label | Libretto | Lyrics | MD | MagneticTape | Medium | Membership | Movement | MusicArtist | MusicGroup | MusicalExpression | MusicalItem | MusicalManifestation | MusicalWork | Orchestration | Performance | PublishedLibretto | PublishedLyrics | PublishedScore | Record | Recording | RecordingSession | Release | ReleaseEvent | ReleaseStatus | ReleaseType | SACD | Score | Show | Signal | SignalGroup | SoloMusicArtist | Sound | Stream | Torrent | Track | Transcription | Vinyl |

Properties: | activity_end | activity_start | amazon_asin | arranged_in | arrangement_of | available_as | biography | bitsPerSample | bpm | catalogue_number | channels | collaborated_with | compilation_of | compiled | compiler | composed_in | composer | conducted | conductor | contains_sample_from | derived_from | discography | discogs | djmix_of | djmixed | djmixed_by | download | ean | encodes | encoding | engineer | engineered | event_homepage | exchange_item | fanpage | free_download | genre | grid | group | gtin | headliner | homepage | image | imdb | instrument | interpreter | ipi | ismn | isrc | iswc | item | key | label | lc | licence | listened | listener | lyrics | mailorder | mashup_of | media_type | medley_of | member | member_of | membership | meter | movement | movement_number | musicbrainz | musicbrainz_guid | musicmoz | myspace | olga | onlinecommunity | opus | origin | other_release_of | paid_download | performance_of | performed | performed_in | performer | possess_item | preview | preview_download | primary_instrument | produced | produced_score | produced_signal | produced_signal_group | produced_sound | produced_work | producer | publication_of | published | published_as | publisher | publishing_location | puid | record | record_count | record_number | record_side | recorded_as | recorded_in | recording_of | records | release | release_status | release_type | remaster_of | remix_of | remixed | remixer | review | sample_rate | sampled | sampled_version | sampled_version_of | sampler | sell_item | signal | similar_to | singer | supporting_musician | tempo | text | time | track | track_count | track_number | translation_of | tribute_to | trmid | upc | want_item | wikipedia |

Individuals: | album | audiobook | bootleg | compilation | ep | interview | live | official | promotion | remix | single | soundtrack | spokenword |

Figure 2.7: The terms used in the Music Ontology (reproduced from the music ontology website)

of music metadata, from high-level information such as artist name and record label, through to background information such as recording studio equipment settings, and down to low-level information such as temporal pitch and audio feature values. Making such rich metadata publicly available has enormous potential benefits in terms of building sophisticated music search applications and cross-discipline data searches (e.g. 'show me all the artists signed to label x based in country y'). What they do not provide us with though, at least not without further processing of the metadata, are any new insights into the nature of the music itself which has been annotated. For example, although we may represent the temporal onsets of all of the pitches present within a certain piece of music, we cannot *infer* from that set of metadata and the associated ontologies alone that the song in question follows the sonata form, or that it

contains four repetitions of a certain musical motif. The great promise of Semantic Web technologies is that we may infer new information for our existing dataset, because we have carefully and accurately defined the relationships that exist between the concepts we are modelling. In Chapter 5 we describe how we have put this promise to the test by locating repeats of perceptually significant patterns within an RDF representation of symbolic music data, using only Semantic Web technologies.

2.3.6 Software

The various components of Semantic Web technologies exist as specifications[24,25,26]. In this section we list some of the software implementations available which allow us to make practical use of the language specifications.

Core RDF APIs

Several software library families exist which implement core RDF functionality such as parsing, graph creation and navigation, serialisation and query. In many cases, these libraries also provide skeleton APIs for extended functionality such as OWL reasoning, the implementations of which are provided by additional libraries. Jena[27] and Sesame[28] are two such core libraries, both implemented in Java, providing core functionality such as the creation and manipulation of RDF graphs, RDF file parsing, and serialisation to multiple RDF formats Both provide abstract ontology and reasoning APIs, with Sesame also providing limited inferencing capabilities (RDF Schema, and RDF Schema and direct type hierarchy inferencing). Similar APIs implemented in other languages exist, for example rdflib[29] (Python) and librdf[30] (C).

[24] http://www.w3.org/standards/techs/rdf
[25] http://www.w3.org/TR/sparql11-query/
[26] http://www.w3.org/TR/owl2-syntax/
[27] http://jena.apache.org/
[28] http://www.openrdf.org/
[29] https://github.com/RDFLib
[30] http://librdf.org/

Triple Stores

Triple stores can be thought of as databases specifically designed only to store RDF triples. TDB[31] is a triple store provided with the core Jena library, providing persistent file-based triple storage and transaction capability. OWLIM[32] is another Java triple store, which works as an addition to either Jena or Sesame. OpenLink Virtuoso[33], another Java implementation, works as a standalone web server.

OWL Reasoners

Some of the RDF triples within a triple store (or graph), may make use of the OWL semantics and rules to express logic (class membership, or set operations, for example). An OWL reasoner is a software component which, given a set of RDF triples, will infer new facts (triples) by applying the specific logic expressed within the current set of triples according to the semantics and rules of the OWL specification. Any resulting new triples are added to the existing graph. Furthermore, multiple OWL profiles exist[34], which specify particular restrictions on the semantics and rules – any particular OWL reasoner should specify its level of conformance with each OWL profile. Inference engines providing various levels of knowledge representation formalism include OWLIM[32], Pellet[35], and Protégé[36] plus the FACT++ or HermiT plugins. A more complete and detailed list is maintained at the W3C Semantic Web website[37].

2.4 Symbolic Music Data Analysis

Given our stated desire to evaluate the utility of Semantic Web technologies with respect to music content analysis (see Chapter 5), we present in the remaining sections of this chapter brief descriptions of some existing methods for discovering perceptually significant components of symbolic music data. We

[31] http://jena.apache.org/documentation/tdb
[32] http://www.ontotext.com/owlim
[33] http://virtuoso.openlinksw.com/
[34] http://www.w3.org/TR/owl2-profiles/
[35] http://clarkparsia.com/pellet/
[36] http://protege.stanford.edu/
[37] http://www.w3.org/2001/sw/wiki/OWL/Implementations

choose symbolic data because, given that RDF data easily lends itself to knowledge representation, and, especially in conjunction with OWL, inferencing, we suggest that the implementation of an algorithm which operates on clean (i.e. score time and pitch) symbolic data will be far more tractable than one which operates on recorded audio (e.g. the signal processing-based segmentation algorithm described by Hargreaves, Klapuri, and Sandler, 2012).

Computational analysis of symbolic music data can be traced back as far as 1949, with Bronson (1949) proposing a method in which IBM punched cards are used to perform queries on a database of British-American folk-tunes. The method involves a considerable amount of initial manual effort, as a punched card must be created for each folk-tune. Bronson considers the most significant elements of a folk-tune to be:

- Range (authentic, plagal, or mixed)
- Modal characteristics
- Time signature
- Number of phrases
- Nature or pattern of the refrain
- Phrasal Scheme (e.g. ABCD, ABBA, ABAD etc.)
- The final – being identical to the tonic or not
- Initial interval between the upbeat and the first strong accent
- Cadential notes of the other phrases of the tune (i.e. other than the first phrase)

These features are tabulated on each card, along with folk-tune identity and a skeletal outline of the first phrase (main stresses, not a full transcription). The final collection of cards then forms a database, which may be queried by sorting according to some desired characteristic, and "picked out and counted with inhuman speed and accuracy". Interestingly, the ability to study racial and national characteristics is considered too, with the proposal that cards be stained in a colour representing a particular national or racial tradition.

An obvious problem when querying symbolic representations of music is that unless the musical key of a pattern is abstracted away from the pitch intervals, then identical patterns which have been transposed in pitch will not be matched. Dillon and Hunter (1982) overcome this by using a representation system wherein the notes from the main octave of the melody are represented by the numbers 1 to 7, and any below or above this are preceded by L or U respectively (denoting either Lower or Upper). Similar melodies are found by matching boolean combinations of necessary numerical sequences and optional variations (e.g. certain notes are permitted to be missing or different from the search pattern). Lemström et al. (1999) combine pitch and duration information from consecutive notes in order to derive *relative* pitch and duration changes between successive notes ("Relative Interval Slope"), which makes their representation both tempo and transposition invariant. The method is only applicable to monophonic music.

2.4.1 String Processing Algorithms

A common method of representing music as symbolic data is to use strings (Stech, 1981; Mongeau and Sankoff, 1990; Ghias et al., 1995; Lemström, 2000; Conklin and Anagnostopoulou, 2001) (a good overview is given in Meredith, Lemström, and Wiggins, 2002), and then to apply string matching algorithms to the task of pattern recognition and discovery. In a string representation, the aspects of each music event considered to be most relevant (typically pitch, onset time and duration) are represented as a triple (or tuple, quad etc., depending on the number of attributes under consideration), e.g.

$$\{pitch, onset\_time, duration\}$$

Each unique triple is then assigned a character from some character set or alphabet, and general string matching techniques from computer science may be applied. The similarity of two passages of music (represented, for example, by string A and string B) is sometimes measured as the 'edit distance' between the two strings – that is, the number of single-character inserts and deletes (and sometimes replacements) necessary to transform string A into string B. Common to all of these methods though is the fact that they are primarily targeted at monophonic data only. Lemström (2000) presents a method which is able to search polyphonic music databases, although only for a monophonic

pattern. Conklin and Anagnostopoulou (2001) utilise multiple 'viewpoints' – i.e. not just (for example) pitch and onset time, but melodic contour, intervals and duration. Two or more viewpoints may be linked to form a composite viewpoint, which does offer the potential of linking multiple channels, however the focus of their technique and results is still very much monophonic.

2.4.2 Pattern Discovery in Symbolic Music Data

The techniques described so far in this section have been concerned with *querying* symbolic music data – that is, searching a symbolic music corpus for occurrences of a specific sequence of notes (or, in some cases, for a similar sequence). A related research area is that of *pattern discovery* – i.e., setting out to discover all occurrences within a corpus of perceptually significant, but initially unspecified, patterns. One method of achieving this (Conklin, 2010) is to examine both a corpus and an anticorpus; looking for patterns which are overrepresented in the corpus as compared to the anticorpus. A likelihood ratio is then used to evaluate the distinctiveness or interest level of each discovered pattern.

The pair of pattern discovery algorithms SIA ('Structure Induction Algorithm') and SIATEC ('Structure Induction Algorithm Translational Equivalence Class') by Meredith et al. (2002) represent music (as well as other types of) data as a multidimensional dataset, and take a geometrical approach to analysis. The use of multidimensional data allows the authors to analyse polyphonic music (i.e. instrument or channel number may be one of our dimensions) as well as multiple facets of music data, for example onset time, offset time, pitch or timbre. Taking a geometrical approach to pattern discovery also allows the detection of repeat patterns of notes which have been shifted in (for example) pitch. Aside from their applicability to musicology, the authors propose that the algorithms could be used for data compression, given that they describe each pattern only once, and then specify the locations of their translated repetitions, rather than the more data-hungry case of repeatedly recording equivalent patterns.

Collins et al. (2010) apply the SIA algorithms and further variations of them to the task of discovering translational patterns in baroque keyboard works, highlighting in the process a problem they refer to as the "problem of isolated membership" – that is, the SIATEC algorithms tend in some cases

to identify patterns containing 'one-off' occurrences of notes as fundamental patterns, when in fact the pattern *minus* the one-off note would be more appropriate. Accordingly, in Collins and Meredith (2013), the algorithm is refined further, such that this time a pattern may only be regarded as fundamental if it is the intersection of more of more super-patterns. In his PhD thesis, Collins (2011) develops the SIA algorithms further and applies them to the task of automated stylistic composition. Other possible applications of symbolic pattern discovery algorithms include the indexing of symbolic music corpora to aid rapid searches, the analysis of grouping and metrical structure, and as an aid to music composition.

We stated in Chapter 1 that one of the purposes of this research is to investigate how amenable Semantic Web technologies are to algorithmic music content analysis, and in Section 2.4 we also made the decision to work with symbolic data for reasons of tractability. The transposition invariance aspect of SIA and SIATEC is extremely useful, given the perceptual significance of chromatic pitch transposition. Consequently, in Chapter 5, we describe the SIA algorithms in greater depth, and present a Semantic Web implementation of them. Additionally, in order that they may be fairly evaluated, many researchers devising symbolic music data analysis algorithms (Collins and Meredith, 2013; Typke et al., 2003; Lemström and Tarhio, 2003; Lubiw and Tanur, 2004; Clifford et al., 2006) state the computational complexity of their algorithms. With this in mind, we also provide performance evaluation metrics and a discussion of the strengths and weaknesses of such an approach.

2.5 Summary

In this chapter we presented evidence that the accuracy of mixed audio-based MIR techniques has reached a plateau over the last few years, and that single instrument-based MIR is almost always superior to multi instrument. We provided more in-depth background on one particular MIR task – that of structural segmentation, in preparation for our multitrack audio-based segmentation experiment in Chapter 4. Given that the results of all MIR tasks are a form of metadata, we discussed the increasing prominence of the Semantic Web as a metadata sharing methodology, and presented examples of its use both generally and within the MIR community. Finally, building upon the assumption

that multitrack audio-based metadata generation leads us closer to the goal of accurate symbolic representations of recorded music, we introduced general symbolic music data analysis techniques as well as the SIA and SIATEC pattern discovery algorithms, operating on multidimensional (and therefore, in the context of music, potentially multichannel) data. Chapter 5 builds upon these foundations with a detailed description and evaluation of a Semantic Web technology implementation of the SIA and SIATEC algorithms.

Prior to that, collecting together the themes discussed so far, we describe in the following chapter a vision for a new MIR paradigm. This provides the wider context for the research presented in subsequent chapters.

Chapter 3

A Vision of a New MIR Paradigm

In this Chapter we describe an over-arching vision for an alternative MIR paradigm, built around the principles of early, studio-based metadata capture, and exploitation of open, machine-readable Semantic Web data.

3.1 The Current Paradigm

Currently, MIR tasks typically belong to a paradigm wherein every specific type of metadata (e.g. beat onsets, note onsets and pitch, structural segments) is generated by performing one particular set of signal processing tasks on a full audio mix. Inevitably, each of these sets of signal processing tasks will entail a large degree of effort directed towards the isolation of the salient parts of the audio from the full mix. Furthermore, there is no consensus on a community-wide metadata format. Under this paradigm then, if we wanted to generate a comprehensive set of multiple different types of metadata for one song, we would run, in isolation from each other, multiple different signal processing algorithms on the same audio mix, many of them potentially repeating similar 'source-separation' type tasks, and our result would be a set of metadata in (potentially) multiple different data formats. The situation is exemplified in Figure 3.1.

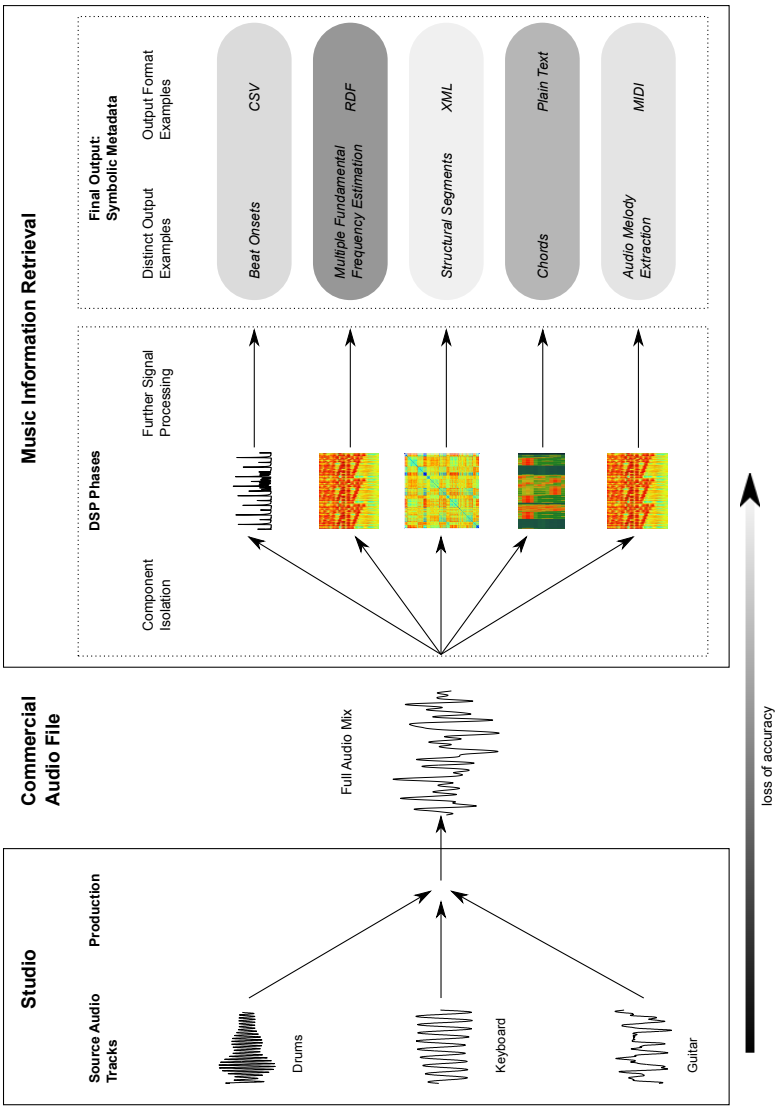

Figure 3.1: Typical MIR metadata generation paradigm

3.2 A New Paradigm

In Chapter 2 we discussed the so-called 'glass ceiling' for MIR algorithm accuracy, the proliferation of the Semantic Web both generally and within the MIR community, and some of the different types of analysis we may perform using symbolic music data. Together, these observations and results point towards a fundamentally different approach to MIR, in which we:

- Exploit the processing power available to us in the recording studio
- Simplify the complexity and/or increase the accuracy of MIR algorithms by targeting source audio tracks rather than the full mix
- Are able to use the results of one MIR algorithm within the execution of another
- Produce a rich set of symbolic, or close to symbolic, metadata for a piece of recorded music
- Exploit the potential of the Semantic Web by publishing our metadata in a common, machine-readable, format
- Infer new musical information at a later date via less computationally expensive processing of our symbolic metadata (e.g. via the use of ontological inferencing or SPARQL queries)

Collectively this forms a larger, over-arching vision, which we will refer to as "Semantic Audio", and which is exemplified in Figure 3.2.

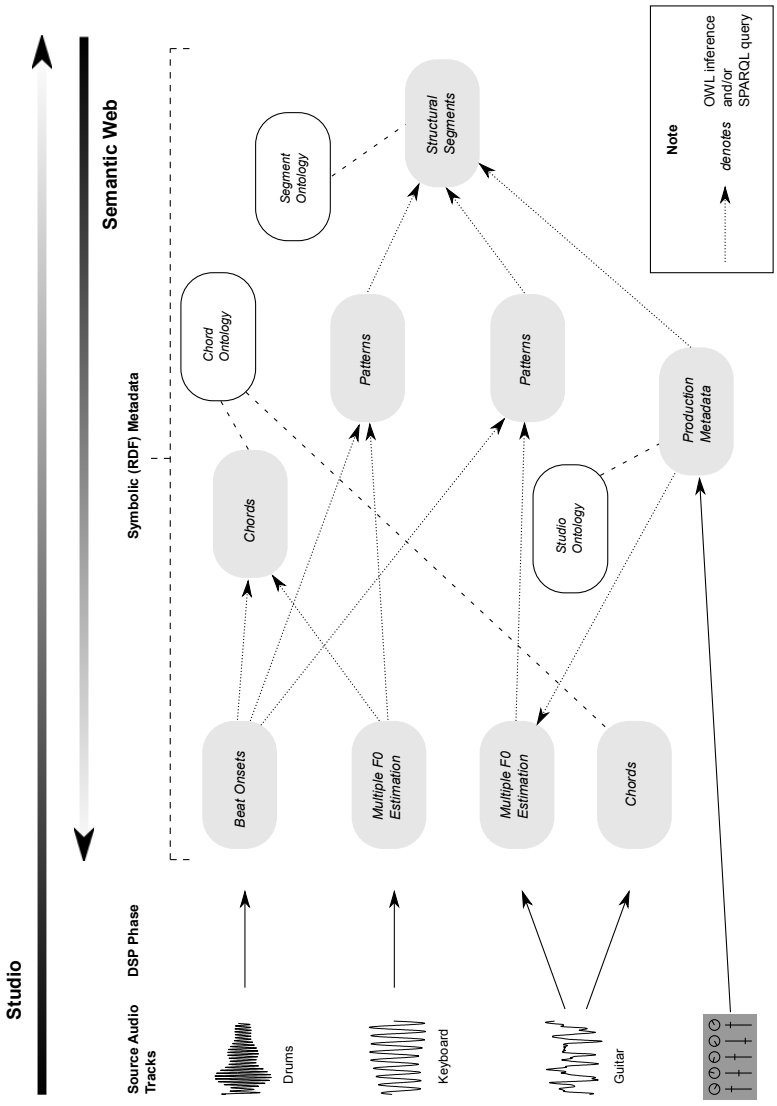

Figure 3.2: Semantic Audio Paradigm

An initial set of accurate symbolic RDF data is generated in the studio – in this example we derive beat onsets from the drum track, multiple F0 data from the keyboard track, chords from the guitar track, studio production metadata directly from the mixing desk and other studio equipment, and another set of multiple F0 data - this time from a combination of the guitar audio track and the studio production metadata. All of this meta and symbolic data can be made available immediately as RDF data on the Semantic Web.

Later, either again in the studio or entirely separately from it, further metadata may be generated via the use of SPARQL queries and/or OWL inferencing – in this example we see the keyboard's chords derived from a combination of the beat onsets and multiple F0 RDF datasets in conjunction with the chord ontology, two separate pattern datasets derived from (a) the keyboard and (b) the guitar multiple F0 datasets, and structural segments derived from these two pattern datasets and the studio production metadata.

As well as raw metadata, Figure 3.2 contains several existing ontologies, for example the chord[1], segment (Fields et al., 2011), and studio (Fazekas and Sandler, 2011; Fazekas, 2012) ontologies. Other, related but not shown ontologies include the music (Raimond et al., 2007), similarity (Jacobson, 2011), and audio features[2] ontologies. In conjunction with one another, these ontologies are the key to enriching our RDF metadata, allowing us to create links between high-level metadata such as artist / track name, through to mid-level data such as structural segments, and down to fine-grained, low-level data such as audio features.

3.3 Use Cases

In this section we present some example use-cases facilitated by this new semantic audio-based MIR paradigm.

3.3.1 Semantic Navigation Around a Multitrack Audio Project

Navigation functionality in present-day Digital Audio Workstations (DAWs) is limited to two forms – manual 'tape recorder' style fast forward and rewind

[1] `http://purl.org/ontology/chord/`
[2] `http://purl.org/ontology/af/`

(wherein the user scrolls backwards or forwards through a visualisation of audio waveforms), and jumps to manually entered labels or intrinsically present boundaries of midi or audio clips. These types of navigation are limited by the extent to which the engineer has labelled pertinent sections, and the degree to which midi and/or audio clips are present. In the case of a multitrack recording of live musicians playing non-digital instruments, the only labels likely to be present will be those entered manually by the studio engineer.

Contrast this to the case of a word processed document, in which paragraphs, sentences, spelling and grammar are automatically highlighted and (if enabled) auto-corrected as the user types the document, or modern digital cameras containing built-in facial recognition software.

By performing structural segmentation in the recording studio (as described in Chapter 4), together with other MIR tasks such as beat tracking and pattern discovery, the DAW's navigation capabilities may be significantly enhanced via the auto-generation of structure, pattern, note and beat annotations. The studio engineer may jump directly to sections such as 'second chorus start' or '3rd beat, 2nd bar of electric guitar verse 2'. Furthermore, as with all the metadata created in the studio, this structure data may be published on the Semantic Web along with artist and track name, which in turn facilitates enhanced navigation for consumers listening to the commercially released full mix.

3.3.2 Custom End-User Audio Content

Besides general music lovers, a subset of consumers exist who welcome the opportunity to be more creative with commercially released music. DJs and remixers for example create new musical works and/or performances from existing ones. By making some or all of the source audio tracks available along with our rich set of symbolic metadata, the end user becomes able to customize their own listening experience, with commands such as 'play chorus minus drums', or 'attenuate keyboard 6dB and repeat verse 3'.

3.3.3 Advanced Online Music Search

Many online music retailers and streaming services provide music recommendations. Regardless of the technique used to generate these recommendations, the search mechanism is firmly in the hands of the service provider – the consumer

has no control over the process.

By making our rich set of studio-generated symbolic data available on the Semantic Web, we facilitate much more sophisticated, user-driven, music search engines. The combination of studio production and melody pattern metatdata, for example, enables a search such as 'Find all Chicago-based quartets with at least one occurrence of the following chord progression'. Possible search criteria include:

- Rhythmic pattern
- Melodic pattern
- Chord progression
- Musician
- Presence/absence of specific instruments
- Era
- Production personal details
- Musical Idiom

Some of these criteria might be difficult to specify, either in terms of the user interface or the typical level of musical expertise of consumers, however the possibility still exists to perform simple queries which compare songs, such as 'Find songs with a similar rhythm pattern to song X'.

3.3.4 Semantic Web Pattern Discovery

Repetition, and therefore pattern discovery, is a key element of music analysis. In the highly cited work "A Generative Theory of Tonal Music" (Lerdahl and Jackendoff, 1996), the authors list "grouping structure", which itself "expresses a hierarchical segmentation of the piece into motives, phrases, and sections", as one of the four fundamental components of their overall theory. Narmour (1992) provides a formal theory of music based heavily upon melodic structure analysis and cognition, whilst Cambouropoulos (1998) considers pattern-matching prior to presenting his String Pattern Induction Algorithm (SPIA) as a key component of his General Computational Theory of Musical Structure.

It follows then that the ability, perhaps retrospectively (i.e. when a musicologist takes an interest in a recorded work some time after the work has left the studio) to discover patterns within the symbolic RDF data originally produced in the studio using our new paradigm (and therefore available on the Semantic Web) would be of great value. The SIA family of pattern discovery algorithms first described in 2002 (Meredith et al., 2002) have spawned much research within the field – see Clifford et al. (2006); Collins et al. (2010); Collins (2011); Collins and Meredith (2013); Collins et al. (2011). In the spirit of open data and our proposed new MIR paradigm, it would be preferable if we could harness the inferencing power of OWL and/or the querying ability of SPARQL in order to implement the SIA algorithms, rather than having to parse our symbolic RDF data for consumption by some programming language that sits outside of the Semantic Web.

3.4 Summary

In this chapter we have described a new paradigm for MIR, in which we perform computationally expensive signal-processing tasks on individual audio tracks in the studio, and publish a rich set of symbolic (or close to symbolic) RDF metadata on the Semantic Web, from which new insights and value may be gained via query and/or inference.

There are many facets to this paradigm, and consequently this is a vast area of research, much of which is beyond the scope of a single thesis. In the following two chapters we start the process by delving more deeply into two of the key themes. In Chapter 4, we explore the question of whether or not there is indeed a quantifiable increase in accuracy when an MIR task is carried out with access to the multitrack audio. In Chapter 5, we show how a pattern discovery algorithm, operating on the type of symbolic data we might hope to be able to generate in the studio, may be fully implemented using only Semantic Web technologies.

Chapter 4

Structural Segmentation of Multitrack Audio

As a first step in our exploration of the new paradigm for MIR outlined in the previous chapter, we describe in this chapter two experiments which apply some of the techniques described in Section 2.2 to the task of identifying the temporal locations of structural boundaries in multitrack audio. We extend the audio feature extraction phase such that features are extracted separately from all of the source tracks present in a multitrack project, rather than the usual case of from a single mono or stereo mixdown audio track.

4.1 Introduction

The manner in which humans listen to, interpret and describe music implies that it must contain an identifiable structure. The terms used to describe that structure will vary according to musical genre, but commonly it is easy for humans to agree upon musical concepts such as chorus, verse, melody, beat, bass, movement, solo, noise and so forth. The fact that humans are able to distinguish between these features implies that the same might also be achieved via signal processing; indeed, over the last few years increases in computing power and advances in MIR have resulted in algorithms which can extract features such as timbre (Aucouturier et al., 2005; Wellhausen and Hoeynck, 2003; Levy and Sandler, 2008), tempo and beats (McKinney et al., 2007), note pitches (Klapuri and Davy, 2006) and chords (Mauch et al., 2009) from polyphonic, mixed source digital music files (e.g. mp3 files, as well as other formats).

A significant problem when attempting to extract features from mixed

source signals is that some complex time or frequency domain signal decomposition must usually be performed in order that the salient parts of a signal may be analysed in isolation; for example a beat tracker will probably need to disregard long term orchestral swells, whilst an algorithm designed to extract melodic information must ignore percussive transients. One related area of research (Fazekas and Sandler, 2007b,a; Fazekas et al., 2008) which avoids this issue, is that of applying the techniques outlined above to the collection of individual source audio tracks available during the recording and/or production stage in the studio.

Sophisticated Digital Audio Workstations (DAWs) are now commonplace, not only in professional recording studios but also in the amateur musician's home studio, enabling consumers to exploit the kind of music recording and production techniques previously only available to a minority who were fortunate enough either to have the budget or opportunity to gain access to expensive studio time. A facility still lacking though is the ability to quickly navigate around the structure of recorded audio. We now take it for granted that we are able to navigate around a word-processed document by character, word, sentence, paragraph, section or chapter, whilst being limited within a DAW to either a fast-forward (or backward) search, a jump to manually entered temporal label, or a manual scroll through the audio. When describing an audio browser for annotation purposes, Tzanetakis and Cook (1999) point out that "The typical 'tape-recorder' paradigm for audio user interfaces is time-consuming and inflexible". The user must rely on audio or visual cues (in the case of examining a waveform display) and his or her own ability to interpret those cues in order to locate a section of interest.

Having access to the original multitrack source audio files theoretically enables us to obtain both a more accurate segmentation, and a richer set of metadata in general, since salient audio features which might otherwise have been occluded to some extent in the mixed version of a song are now able to exert greater influence in our analysis.

It is already possible, using various methods (described in Section 2.2) to structurally segment mixed polyphonic music to a certain extent. The aim of the experiments described in this chapter is to demonstrate an improvement in segmentation accuracy when multitrack rather than mixed audio data is analysed. In the first experiment (Section 4.4), we achieve this by applying one

particular segmentation technique to multitrack audio and comparing the results against (a) results obtained from the same technique applied to mixed versions of the same songs, and (b) results obtained using a state-of-the-art segmentation algorithm (Mauch et al., 2009), again applied to the mixed versions. In the second experiment (Section 4.5), we modify our method such that the instrument type of each individual source track is taken into account. The potential applications of such techniques are manifold, and include improved synchronisation of audio clips across multiple tracks, segment-specific application of audio effects, improved comparison of recording takes, and general editing and navigation tools.

The particular way in which one would structurally segment music is closely tied to the musical genre under consideration. Intuitively, rock and pop music seems like one of the more unambiguous types of music for humans to segment (compared to classical or improvisational jazz, for example) due to the common repetition of melodic phrases, chord progressions and beats. These musical entities are typically repeated every few bars, and sequences of these bars themselves form verses or choruses. In reality though, whatever the genre, it is in the very nature of music that rules exist to be broken, and so we should never rely too rigidly upon assumptions about metrical structure, chord progressions, time signatures or anything else. Consequently, these experiments will focus mainly on rock and pop music (we include in this definition genres such as soul, R 'n' B, blues, dance, latin pop, easy listening, folk and electronica). Classical music and jazz will not be considered.

The rest of this chapter is set out as follows; in Section 4.2 we present a hypothesis, followed by a description of a new multitrack audio dataset in Section 4.3. Our first method of segmenting multitrack audio, based on combined, weighted audio features, is described in detail in Sections 4.4 and 4.4.1, and the corresponding evaluation technique is described in Section 4.4.2. Results are stated in Section 4.4.3, followed by a discussion in Section 4.4.4. In Section 4.5 we describe our second segmentation method, this time, one which selects audio features based upon the instrument type present in each individual source audio track. The results of this second experiment are presented in Section 4.5.4, with a discusscussion in Section 4.5.5. Finally, we present our intermediate conclusions in Section 4.6.

4.2 Hypothesis

Commercially recorded music usually starts life in the studio as a multitrack recording before being mixed down to stereo during the production phase. There are exceptions; live performances, especially of classical or jazz music, might be recorded using 'ambient' rather than close miking techniques for example. We concern ourselves here with the former type of recording (see Huber and Runstein, 2005, for a general overview of recording techniques). Typically, multitrack recordings will have somewhere between eight and 24 tracks, although there is no hard upper limit. Each track is usually a recording of a single instrument or voice, although in some cases (for example drum kits, string sections or choirs), there might intentionally be multiple sound sources recorded on to a single track, or unintentionally in other cases due to microphone 'bleed'. For the duration of a recorded piece of music, some of these individual sources might be producing little or no sound. The temporal changes in activity of individual instruments is potentially lost to a certain degree in the final mix, and our hypothesis is that having access to the multitrack version of a recording enables us to avoid this loss of relevant information by calculating features from all of the individual source tracks, rather than just the final mixdown as is usually the case.

4.3 Multitrack Audio Dataset with Structural Segment Annotations

In order to evaluate any music segmentation technique, a test set of human-annotated musical audio is required. Several already exist[1,2] for fully mixed audio, but no ground truth annotations for multitrack audio projects exist. To that end, we have created a new, publicly accessible[3] multitrack audio dataset consisting of 104 pop and rock songs.

[1] http://www.cs.tut.fi/sgn/arg/paulus/beatles_sections_TUT.zip

[2] http://www.ifs.tuwien.ac.at/mir/audiosegmentation/dl/ep_groundtruth_excl_Paulus.zip

[3] http://c4dm.eecs.qmul.ac.uk/rdr/handle/123456789/36

4.3.1 Selecting and Obtaining Audio

When compiling a ground truth annotated audio dataset, it is important to ensure that the material is both suitable for its intended purpose, and also not subject to copyright restrictions. We stated in Chapter 1 that we are limiting the scope of our segmentation experiments to rock and pop music only, so we require multitrack versions of songs from those genres. The multitrack audio projects which form our test set for Chapter 4 come from a number of sources; the Creative Commons 'ccMixter' website[4], artist websites, donations from friends and colleagues, and a commercial karaoke song website[5], providing individual audio tracks including vocals from cover versions of popular western songs. The audio from the karaoke song website unfortunately is not free of copyright restrictions, however, at the time of writing, the cost of each song was very small. All other audio material obtained is sharable according to the terms of any applicable license, details of which are provided with the dataset.

4.3.2 Annotation

Annotation of our dataset was carried out by two musically trained undergraduates according to the guidelines set out in the Structural Analysis of Large Amounts of Music Information[6] (SALAMI) project. In brief, these guidelines describe conventions for annotating high-level musical structures such as intro, chorus and verse, as well as mid-level structures such as a melodic phrases or chord progressions spread over a small number of bars.

Our dataset annotations include segment labels as well as temporal boundaries, although for the purposes of our own segmentation experiments we only make use of the temporal boundaries.

Ambiguities

Before setting out the instructions given to our annotators, it is worth discussing some of the ambiguities inherent in such a task. When describing structural segmentation, we frequently use terms such as chorus, verse, and bridge. Some

[4] `http://ccmixter.org/`

[5] `http://www.karaoke-version.co.uk/`

[6] `http://ddmal.music.mcgill.ca/research/salami/annotations`

dictionary definitions of these terms follow:

verse

noun

"a group of lines that form a unit in a poem or song; a stanza: the second verse."

chorus

noun

"a part of a song that is repeated after each verse, typically by more than one singer."

bridge passage

noun

"a transitional section in a musical composition leading to a new section or theme."

Whilst making a reference to a song, the definition of 'verse' above is primarily defined in terms of 'lines', i.e. from a poem. The definition of 'chorus' tells us that it is "a part" of a song, but does not define 'part' in any more detail, such as time in seconds, or length in number of beats or bars. It is repeated after a verse, however we do not have a solid definition of what a verse is. A 'bridge passage' is "a transitional section... leading to a new section or theme" – does this mean that all sections preceding a new section or theme are bridges, or only some? If only some, how do we make the distinction?

Despite this, these terms are in such common use that it would seem perverse to claim that they are too ill-defined for us to ask anyone to identify where they occur within a song. Ultimately these ambiguities mean that this task is unavoidably subjective, however that is also the reason why we need human generated annotations – in the absence of a precise definition of what constitutes a structural segmental, the best reference we have is human judgement, and that judgement is what we would like our algorithm to replicate. Peeters and Deruty (2009) offer a good discussion regarding the robustness of segmentation evaluation techniques, whilst Bruderer et al. (2006) demonstrate that despite the subjective nature of music segmentation, there is a correlation between the number of subjects identifying a particular boundary and the level of salience

attached to it.

Instructions to Annotators

Annotators are asked to identify large-scale, musically similar segments, where similarity may be rhythmic, melodic, or harmonic. The temporal boundary, in seconds, between each of these segments must be identified, and a label applied to each segment, with similar segments receiving the same label. The label must be an uppercase letter, e.g. A, B, C etc., with Z being used for 'special' sections, such as speech or applause. The prime symbol may be applied to a segment label if the annotator judges it to be a significant variation of some other segment, such a transposed melody.

Lowercase letters should be applied to smaller scale, similar segments, but not at the level of individual notes. Every segment labelled with an uppercase letter must also be labelled with a lowercase letter, but the converse is not true. Lowercase labelled segments must share similarity across larger segments – i.e. a small segment labelled Aa must be similar to Ba (even though on the larger scale, A is not similar to B). No segment should be unlabelled (silence counts as a segment).

Optionally, musical function labels from a controlled vocabulary may be applied to segments. The words in the vocabulary (see the SALAMI guidelines[7] for more details regarding the choice of these words) are:

bridge	**chorus**	**coda**	**end**
fadeout	**instrumental**	**interlude**	**intro**
main theme	**outro**	**pre-chorus**	**pre-verse**
silence	**solo**	**(secondary) theme**	**transition**
verse			

Finally, and again optionally, the name of the lead instrument may be applied to any segment. There is no controlled vocabulary for this. Each boundary time, uppercase label, lowercase label, musical function label and leading instrument name must be separated by a comma. Parentheses may be used to show that

[7] `http://ddmal.music.mcgill.ca/research/salami/annotations`

a leading instrument persists across several segments. An example annotation, for the song 'Dreams' by 'Another Dreamer' is shown below:

```
0.000000000,silence
0.938775510,A, a', pre-verse, (vocal
17.729886621,B,b, pre-chorus
26.092108843,b
34.432653061,c, chorus
42.811700680,c'
51.118367346,C, d, verse
59.481632653,d
67.567346938,e, bridge
69.915283446,A, a, pre-verse
78.251247165,a
86.599931972,B, b, pre-chorus
94.950702947,b
103.302040816,c, chorus
111.654081632,c'
119.977505668,C, d, verse
128.360997732,d
136.434648526,e, bridge
138.792562358,A, a, pre-verse
147.117278911,a
155.465170068,B, b', pre-chorus
163.829478458,b'
172.189206349,c', chorus
180.538412698,c'
188.852290249,C, d, verse
197.221950113,d, vocal)
205.313015873,e', outro
209.983287981,silence
210.786213151,end
```

Dataset Statistics

The structural segment annotated multitrack audio dataset consists of 104 songs, with a total of 3119 ground truth structural segmentation boundaries, an average of 9 tracks per song (minimum 4, maximum 17), and an average song duration of 3 minutes 56 seconds (minimum 1 minute 36 seconds, maximum 10 minutes 3 seconds). The ground truth annotations which accompany them took approximately 280 man-hours to create.

Dataset Contents and File Format

The files available in the Research Data Repository[8] provide ground truth structural segmentation annotations and corresponding mutltitrack audio recordings. The multitrack audio recordings are provided in one of two forms:

1. Actual audio files, contained in three zip archives
2. Hyperlinks to commercially available multitrack audio

The ground truth structural segmentation annotations are provided as comma-separated value (csv) files. The format of these csv files complies with the 'Structural Analysis of Large Amounts of Music Information (SALAMI)' guidelines, which are described in the file SALAMI Annotator Guide[9]. These guidleines were produced by McGill University.

In the wider context of this thesis, and in particular the use of RDF data outlined in Chapter 3 and used in Chapter 5, it could be argued that this dataset should be made available as RDF data. However, the overriding purpose of the dataset is to serve as test data for the multitrack versus mixed audio segmentation experiments described in this chapter, and also for other researchers to use. For the sake of simplicity and because of the existence of other similar datasets also using the SALAMI guidelines, we have elected to publish the annotations as csv files only at the present time, although we may augment them with RDF versions in the future.

[8] `http://c4dm.eecs.qmul.ac.uk/rdr/handle/123456789/36`

[9] `http://ddmal.music.mcgill.ca/research/salami/annotations`

Files in the Repository

- `multitrack_audio_1.zip`, `multitrack_audio_2.zip`, and `multitrack_audio_3.zip` – archived multitrack audio files, where the first part of each folder name reflects the artist and song names. Audio files are either in mp3 or wav format.
- `commercial_audio_files.txt` – a text file containing hyperlinks to commercially available multitrack audio files.
- `structural_segments.zip` – ground truth structural segmentation annotations, where the first part of each filename reflects the artist and song names, and ends with '`_gt`' denoting ground truth.
- `licenses.txt` - a text file stating which type of license applies to which song.
- `SALAMI-Annotator-Guide.pdf` - a description of the format of the structural segmentation annotations in the csv files

Availability

All of the annotations and audio files, apart from the commercial karaoke material (which may instead be purchased), are available for download[10], and the author would like to encourage other researchers to make use of them.

Comparison of Annotation Styles

Unintentionally, in two instances, our two human annotators both annotated the same song. Usefully though, this provides us with an opportunity to compare and contrast (at least in two cases) the segments chosen by the two annotators. Figure 4.1 shows the two alternative segmentations for the song 'Armistice' by Phoenix, which is a pop song in 4/4 time (the alternate colour shades used in the images are purely to aid visualisation of consecutive segments – they do not imply anything about the nature of the segments).

[10] `http://c4dm.eecs.qmul.ac.uk/rdr/handle/123456789/36`

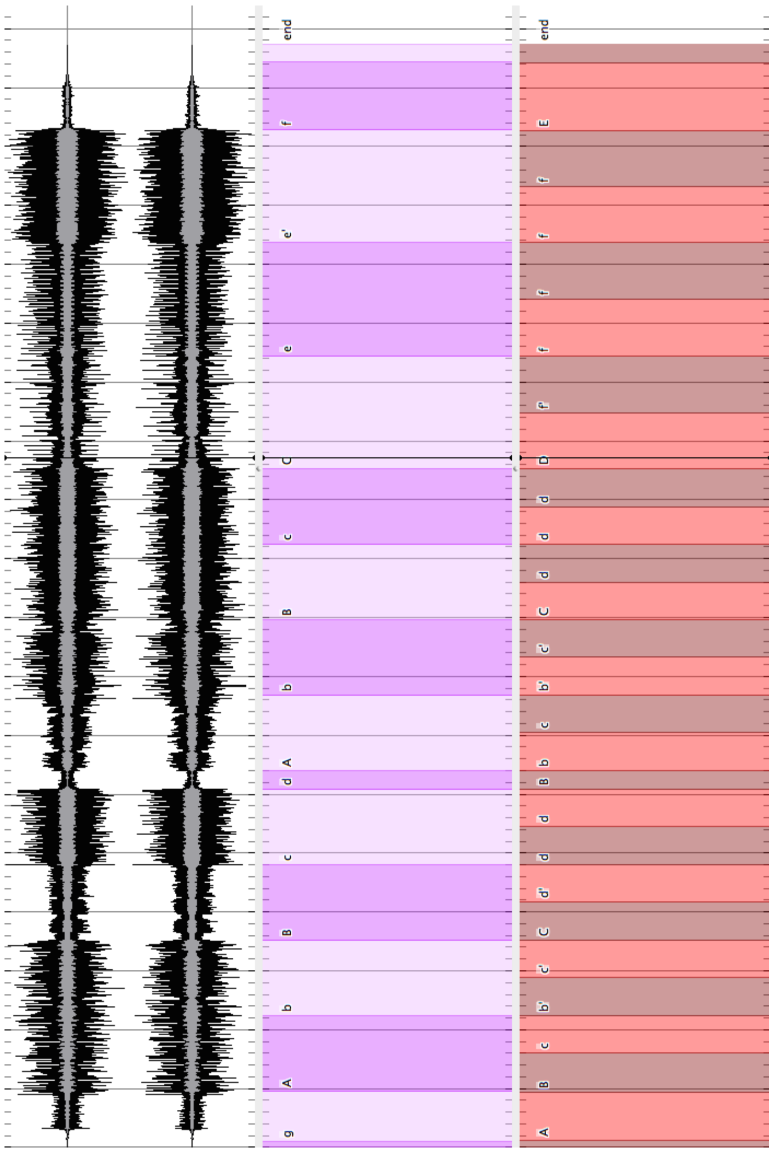

Figure 4.1: Two alternative segmentations of the song ‘Armistice’ by Phoenix. The top pane shows the audio waveform of the song, the middle pane shows the segmentation chosen by annotator A, and the bottom pane shows the segmentation chosen by annotator B.

Boundary retrieval F-values, as described by Levy and Sandler (2008), provide us with a quantitative measure of the similarity of the two annotations. More commonly used to compare experimentally derived boundaries with human 'ground truth' boundaries, we use them instead here to compare two human annotations, one of which we must arbitrarily nominate as the 'reference'. The metrics are defined as follows:

$$P = \frac{|P_r \cap P_h|}{|P_r|} \tag{4.1}$$

$$R = \frac{|P_r \cap P_h|}{|P_h|} \tag{4.2}$$

$$F = \frac{2PR}{P + R} \tag{4.3}$$

where P is boundary retrieval precision, R is boundary retrieval recall, F is boundary retrieval F-value, P_r is the reference set of segment boundaries identified by one human annotator, and P_h is the set of segment boundaries identified by the second human annotator. Picking (arbitrarily) annotator B's annotation as the reference, we calculate the boundary retrieval F-values for a +/- 0.5s tolerance as:

$$P = 1$$

$$R = 0.61$$

$$F = 0.76$$

Annotator A has chosen five high-level (indicated by upper-case letters) segments, whilst annotator B has chosen seven. On three of these occasions, both annotators are in exact agreement. Ignoring the particular labels applied, annotator B has chosen all the boundaries that annotator A has, plus some additional ones of their own. The mappings between the uppercase labels used by the two annotators are consistent, albeit in one case annotator B has judged the high level segment to start one low-level segment earlier than annotator A. Ignoring the identical intro and outro segments (labelled 'g' and 'f' respectively in the middle pane), annotator A consistently subdivides their high-level segments into two lower-level segments of either eight or twelve bars. In each of these cases, annotator B elects to subdivide the segments twice as often, i.e. into four segments of either four or six bars. This explains the perfect precision

score, along with the lower (but still good) recall score. It is hard to say which annotator is right or wrong in this respect – these are (arguably) just different levels of a hierarchical segmentation (indeed, it is the annotator's lowercase labels which are in disagreement here, not, generally, the uppercase ones).

The two segmentations for the song '1901', also by Phoenix, are displayed in Figure 4.2. Picking annotator B's annotation as the reference again, the boundary retrieval F-values for this example (using a +/- 0.5s tolerance) are:

$$P = 0.90$$

$$R = 0.66$$

$$F = 0.76$$

This time annotator A has chosen seven high-level (upper-case labelled) segments, whilst annotator B has chosen eight. On seven of these occasions, both annotators are in exact agreement. The mappings between the uppercase letters used on these seven occasions are consistent. A similar trend to the previous example can be seen – in the majority of cases annotator B again subdivides high-level segments twice as often as annotator A, although there are occasions here when both annotators choose identical low-level segments, and one occasion when it is annotator A who subdivides a high-level segment twice as much as annotator B. There are also occasions in both examples when the two annotators disagree on the location of a high-level boundary, although this is rare.

It is clear from these observations that given the same instructions, different people group the same metrical structures slightly differently. This will inevitably place an upper limit on the level of accuracy possible from algorithmically generated segment boundaries, although we have already acknowledged the intrinsic ambiguities of such a task. We might mitigate these ambiguities by insisting upon multiple human annotations of every song, and then deriving a single human annotation via some form of boundary vote count, however there are practical difficulties in obtaining so many human annotations for a large dataset.

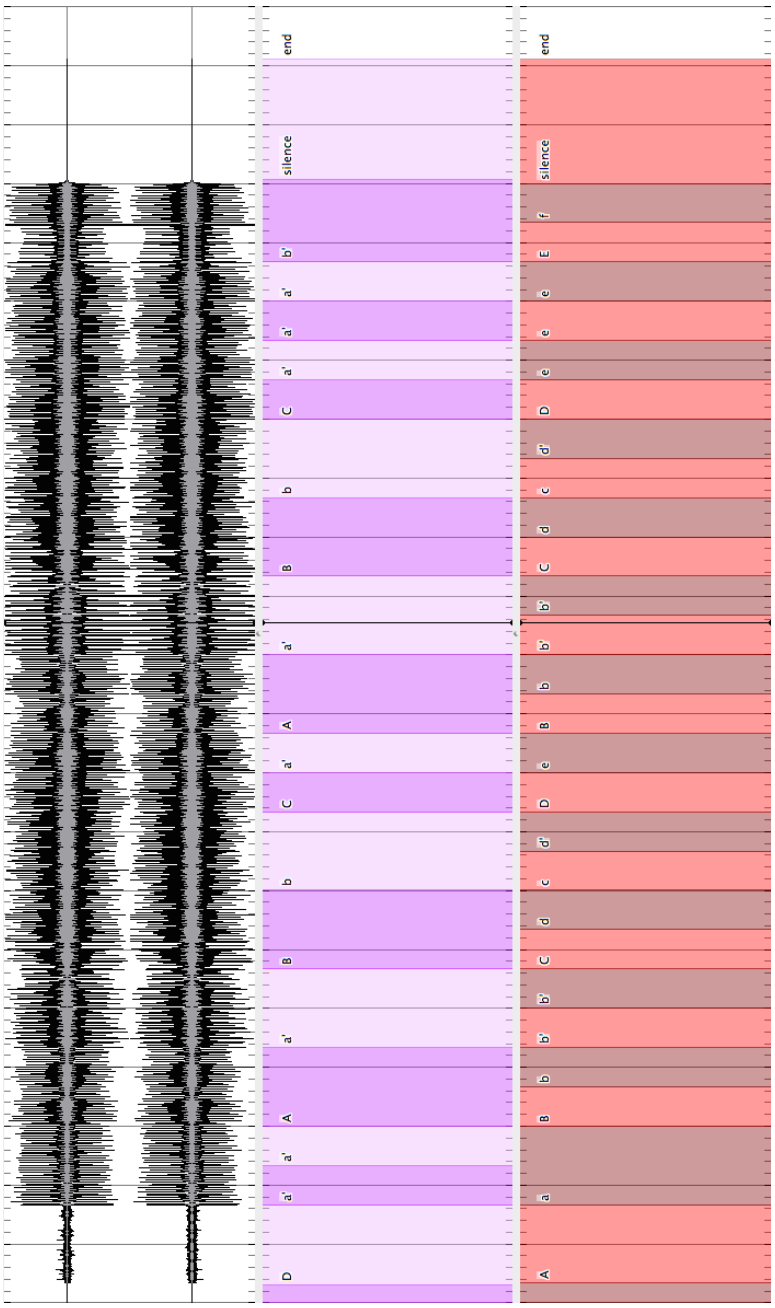

Figure 4.2: Two alternative segmentations of the song '1901' by Phoenix. The top pane shows the audio waveform of the song, the middle pane shows the segmentation chosen by annotator A, and the bottom pane shows the segmentation chosen by annotator B.

It is important to bear in mind that a comparison of two human annotations for each of two songs is not statistically significant, hence we must not draw any solid conclusions from these comparisons. Any attempt to further dictate the rules given to the annotators runs the risk of placing undue bias on the dataset. Rather, we accept that our dataset inherently reflects the ambiguities of structural segmentation, and we take this into account when using the dataset to evaluate any experimental results.

4.4 Combined and Weighted Audio Feaures

If we are to use some of the segmentation techniques described in Section 2.2, we must determine which audio features to extract from our audio files, and in what proportions to use them when calculating self-distance matrices. Hence, our first experiment extracts four common but disparate types of audio feature from every audio track, and performs repeat segmentation calculations using different weightings of all four features, in order to find the optimum feature weights. We employ n-fold cross-validation of our dataset in order to ensure a distinction between training and test data, and therefore avoid over-fitting.

4.4.1 Experimental Method

Our method starts, as is common in segmentation tasks, by calculating audio features for frames of audio which are time-aligned to beats. Beat tracking algorithms are capable of analysing single channel (or stereo) audio files and producing lists of predicted temporal beat locations; in our case though we have multiple audio channels, so we perform a simple mixdown step first. In the absence of an "official" version of the final mix we simply sum the individual source tracks and normalise. We then use the beat tracking method described by Ellis (2007) to find the beat locations within this simple mixdown audio file (Figure 4.3). The mixdown file is used purely for finding beats, we now disregard it and return to consider the multitrack audio files.

As described in Section 2.2, there are several different types of audio features we could choose to extract and analyse. At this stage we do not know which will produce the most effective results, although in the case of final mix audio several authors have carried out investigations in order to determine the relative merits of the different types of audio features with regard to segmentation. Paulus and

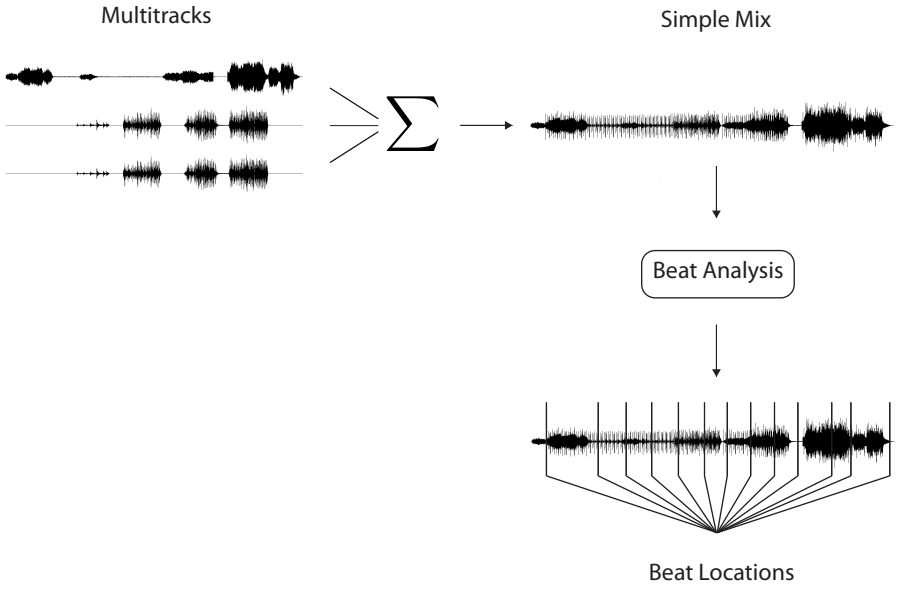

Figure 4.3: Extracting beats from a simple mix of multitrack audio

Klapuri (2008) conclude that either MFCC or chroma features alone work well, however, they worked with final mix audio.

Bruderer et al. (2006) conclude that changes in timbre, changes in level, repetition, and breaks/pauses provide strong cues for the perception of structural boundaries in music. Timbre has been successfully modelled by Aucouturier et al. (2005) using MFCCs in conjunction with Gaussian Mixture Models, whilst a simple measure of RMS energy will provide a measure of signal level, including breaks and pauses. Jensen et al. (2005) reported good results when using the rhythmogram to segment popular chinese music.

It is instructive to examine some example self-distance matrices derived using these features, in order to gain an intuitive grasp of how well each one is able to model the structural changes in some example songs. Figure 4.4 shows the self-distance matrix images obtained using MFCCs (left), and chroma (right) features, both for the same song, "Sunrise" by Shannon Hurley. Musically, the song is a fairly traditional sounding pop ballad, containing a typical verse/chorus/bridge structure, with strong melodic content and percussion. The MFCC-derived image shows both a very clear block structure at an appropriately large timescale, as well as very definite stripes (e.g. in the region between approximately 140s and 180s on both axes) indicating repetition of a sequence of timbral changes. It seems likely in this case that further analysis

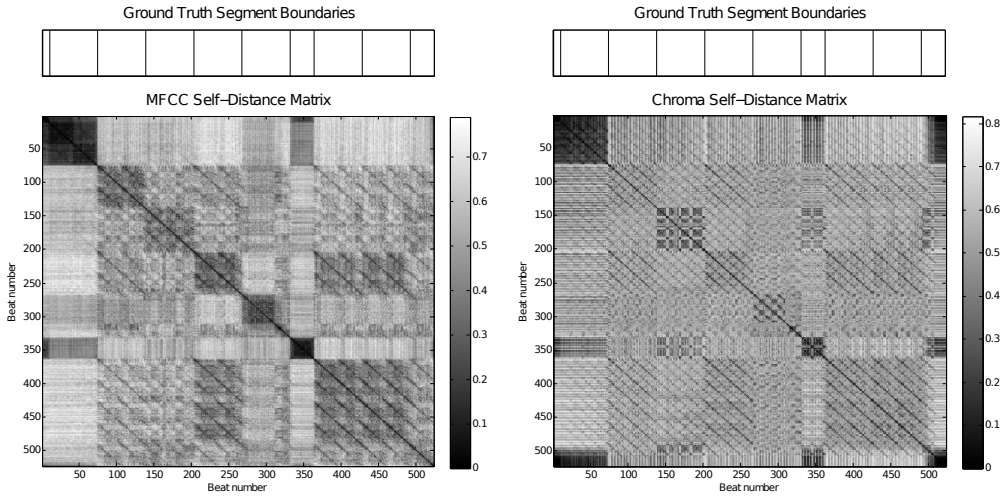

Figure 4.4: Self-distance matrix images for "Sunrise" by Shannon Hurley

could provide us with plausible segment boundaries. Conversely, whilst the same block structure is again evident to a certain degree in the chroma-derived image, the detail is less well-defined; the image has an overall fuzziness, possibly implying that further analysis will be less likely to reveal the structure we're seeking (albeit the stripes are still strongly evident). By way of comparison, Figure 4.5 shows the RMS energy-derived (left) and chroma-derived (right) self-distance matrix images for the song "Hyperpower" by Nine Inch Nails. This song is overwhelmingly defined by changes in timbre and dynamics, on both small and large timescales, with little to no harmonic variation. Accordingly, the RMS energy-derived image exhibits a particularly clear block structure on a large timescale (∼20s), reflecting the obvious level differences between each structural segment, whilst the chroma-derived image is slightly less well defined on the larger timescale, but does show good definition over smaller time periods (∼1s).

We surmise from the similarities (and dissimilarities) evident in Figures 4.4 and 4.5 that all three of these audio features provide relevant, yet different, insights into the structure of a piece of music. For completeness, we also add the rhythmogram feature to this list. We will run multiple experiments using different weightings of these four features in order to determine the most effective combination.

These audio features are calculated for each (beat-aligned) frame of audio

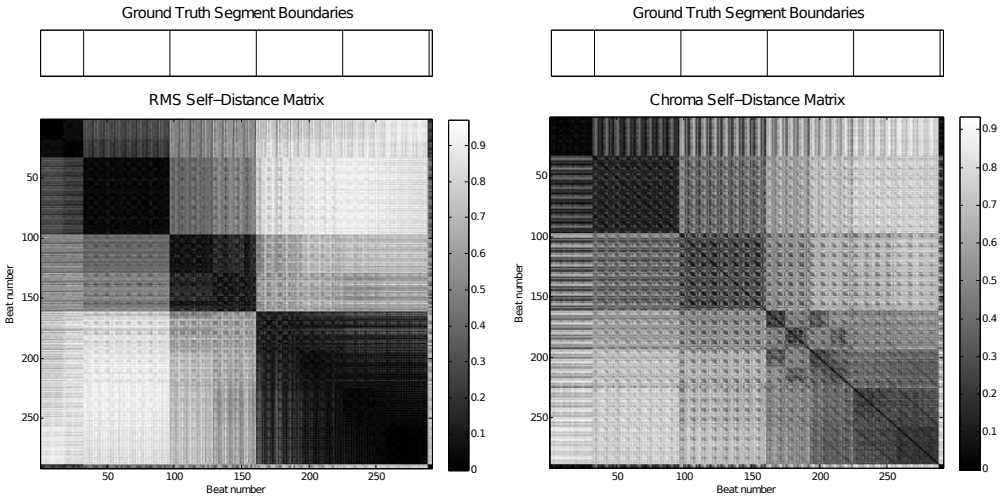

Figure 4.5: Self-distance matrix images for "Hyperpower" by Nine Inch Nails

from each individual track. In each case the RMS energy feature is a single number, MFCC is a thirteen element vector (each frame is characterised by 13 cepstral coefficients), the chroma feature is a twelve element vector (each element indicating a measure of the level of one of the 12 chromatic pitches present in that audio frame), and the rhythmogram feature is a two hundred element vector, representing the rhythmic change at 10ms steps over a 2s window.

For MFCC feature calculation, we use Ellis'[11] Matlab function with a 25ms window length, 10ms hop time, Mel filter band edges at 0Hz and 20000Hz, 40 warped spectral bands, type 2 discrete cosine transform type, and no liftering, preemphasis filter, or dither. For chromagram feature calculation, we use the LabROSA-coversongID[12] Matlab code with 4096 Fast Fourier Transform (FFT) length.

The perceptual spectral flux component of the rhythmogram feature is calculated using a step size of 10 ms and a block size of 46 ms. An equal loudness contour with a phon value of 60 was found empirically to work best for the frequency weighting W (where W is the frequency weighting used to represent an equal loudness contour – see Equation 2.10). The rhythmogram itself is then calculated using a 10ms step size and summing window length of 50 frames, before being beat-synchronised.

[11] `http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/mfccs.html`
[12] `http://labrosa.ee.columbia.edu/matlab/chroma-ansyn/chromagram_IF.m`

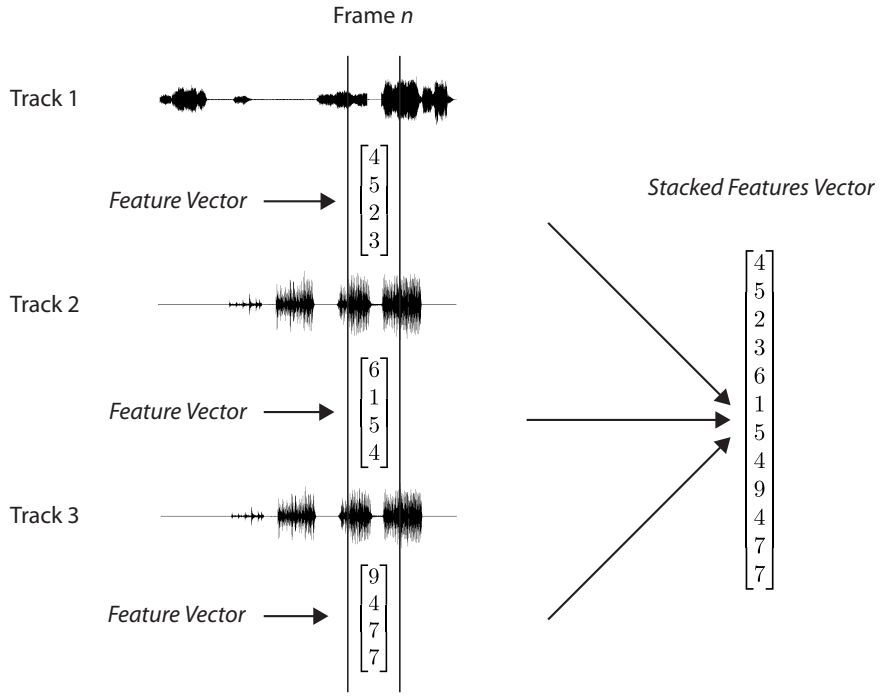

Figure 4.6: Stacking the feature vectors obtained from multiple audio source tracks

For each frame, we stack all of these features into a single vector. For an 8 track case this vector would be 1808 elements long; 13 MFCC values, 12 chroma values, 200 rhythmogram values and one RMS energy value for each of the eight tracks. In the case where we are analysing the final mix only, the feature vector would be 226 elements long. We then apply one of the feature weighting combinations under investigation to these vectors; an example would be to multiply the MFCC features by 100, the rhythmogram features by 10, the RMS feature by 10 and leave the chroma feature untouched. The full set of weightings under consideration is formed by varying the relative weights of each feature by 1, 10, and 100 with respect to every other feature, resulting in 65 different configurations, and these weightings are applied after the individual features have been normalised using z-score.

The resulting collections of vectors calculated for all audio frames then form the sets of input data for our self-distance matrix calculations. Figure 4.6 illustrates the technique for a hypothetical situation in which there are three audio tracks, each producing a four element feature vector, and for one particular weighting combination.

The self-distance matrix is calculated from these feature vectors using Equation 2.12, and we then calculate the standard novelty score (an example is shown in Figure 2.5) using Equation 2.15, with a kernel size of 32. As stated in Section 2.2.4, we evaluate two different methods of picking peaks from the novelty score. In the first method, our segment boundary locations are simply selected as the locations of all those peaks in the novelty score whose height exceeds some scaled factor of the average peak height for each song; the precise value of this scaling factor must be learnt, and we try factors of 0.5, 0.75, 1, 1.25, 1.5, 1.75 and 2 times the average peak height. Hence, using this method, we obtain seven different candidate sets of segment boundaries for every song. There is certainly scope to improve on this simple method though, and consequently our second method (Brennan, 2010, described in Appendix A) is more complex, utilising low-pass filtering in order to remove the presumably irrelevant small-scale peaks from the novelty score. This method produces three candidate sets of segments boundaries. We present the results of both methods in order to demonstrate the potential advantage of fine-tuning the peak-picking method, however we do not claim that either method is optimal. Rather, we concentrate on demonstrating that in either case, there is an advantage to be gained from using multitrack rather than mixed audio.

As a final step, given the knowledge that the SALAMI annotation guidelines (see Section 4.4.2) dictate that the very start and end of the song are marked as boundaries (even if they consist of periods of silence), we check whether or not the algorithm has picked out these locations and if not, we add boundaries there.

4.4.2 Evaluation

Our experimental results consist of either seven (in the case of the simple novelty score peak picking method) or three (when using the Brennan, 2010, method) alternative segmentations for every feature weight configuration applied to every song. We employ n-fold (where n=4 in our case) cross-validation to determine the boundary retrieval F-value, precision and recall of every set of n training songs, and for each feature weighting and segmentation level configuration. Boundary retrieval F-values are calculated as described by Levy and Sandler (2008), by comparing our experimentally derived segment boundaries against

the ground truth data using

$$P = \frac{|P_m \cap P_h|}{|P_m|} \tag{4.4}$$

$$R = \frac{|P_m \cap P_h|}{|P_h|} \tag{4.5}$$

$$F = \frac{2PR}{P + R} \tag{4.6}$$

where P is boundary retrieval precision, R is boundary retrieval recall, F is boundary retrieval F-value, P_m is the set of segment boundaries identified experimentally and P_h is the set of segment boundaries identified by a human annotator.

This allows us to select an optimum feature weight and segmentation level configuration for each training song set. We then take the test song segments derived using the optimum parameter configurations for the corresponding training group set, group them into one collection of segments (i.e. as if all 104 songs formed one long song), and calculate the final F-value, precision and recall values by comparison with the ground truth data. The optimum weight and segmentation level configurations are deduced by taking the average of the n optimum training set configurations. This process is applied to:

1. Segments derived using our method applied to the multitrack data
2. Segments derived using our method applied to the single-channel mixed data

F-values are also calculated from the segments derived using Mauch et al.'s state-of-the-art[13] method (Mauch et al., 2009) applied to the single-channel mixed data. When comparing segment boundaries against ground truth boundaries, tolerances ranges of 1s (+/- 0.5s) and 3s (+/- 1.5s) were used.

4.4.3 Results

The boundary retrieval F-values for the complete set of segments obtained from the 104 song test set, obtained using the three different methods (Mauch applied

[13]Ranked first in the 2009 MIREX Music Structure Segmentation Task

Experimental Method	1 second tolerance			3 second tolerance		
	F-value	Precision	Recall	F-value	Precision	Recall
Mauch, applied to mixdowns	0.29	0.44	0.22	0.43	0.65	0.32
Hargreaves, applied to mixdowns (simple peak pick)	0.29	0.29	0.30	0.51	0.43	0.63
Hargreaves, applied to multitracks (simple peak pick)	0.35	0.33	0.38	0.56	0.50	0.64
Hargreaves, applied to mixdowns (Brennan's peak pick)	0.30	0.29	0.31	0.53	0.51	0.55
Hargreaves, applied to multitracks (Brennan's peak pick)	0.38	0.37	0.39	0.60	0.57	0.62

Table 4.1: Segment boundary retrieval comparisons with ground truth data, using combined and weighted features

to mixdowns, Hargreaves applied to mixdowns, and Hargreaves applied to multi tracks), and for both peak-picking methods (simple and Brennan's), are shown in Table 4.1.

The F-value figures achieved when analysing full multitrack data show significant improvement when compared to the results for mixed data, regardless of whether our own or Mauch's algorithm is used. The greatest improvement is achieved using our method together with Brennan's peak-picking algorithm. Figure 4.7 shows the corresponding optimum feature weight configurations used to generate the F-values in Table 4.1 (these values are obtained by averaging the four different optimum configurations found during four-fold cross-validation). For clarity, we omit the configurations derived when using the simple peak-picking method from this Figure. In all cases, the rhythmogram feature offers very little benefit to the segmentation; this result is consistent with earlier findings by Paulus and Klapuri (2008). When full use of the multitrack data is being made (the 2 leftmost sets of results), an equal weighting of chroma, RMS energy and MFCCs is found to be optimal, whilst when fully mixed data is used (the 2 rightmost sets of results) the importance of the RMS energy and chroma features declines to certain extents depending upon which tolerance level is under examination. It is important to note though that for reasons of tractability, we limited ourselves to relative weightings of 1, 10 and 100 – if more combinations had been tested it is less likely that any optimum weightings would have been

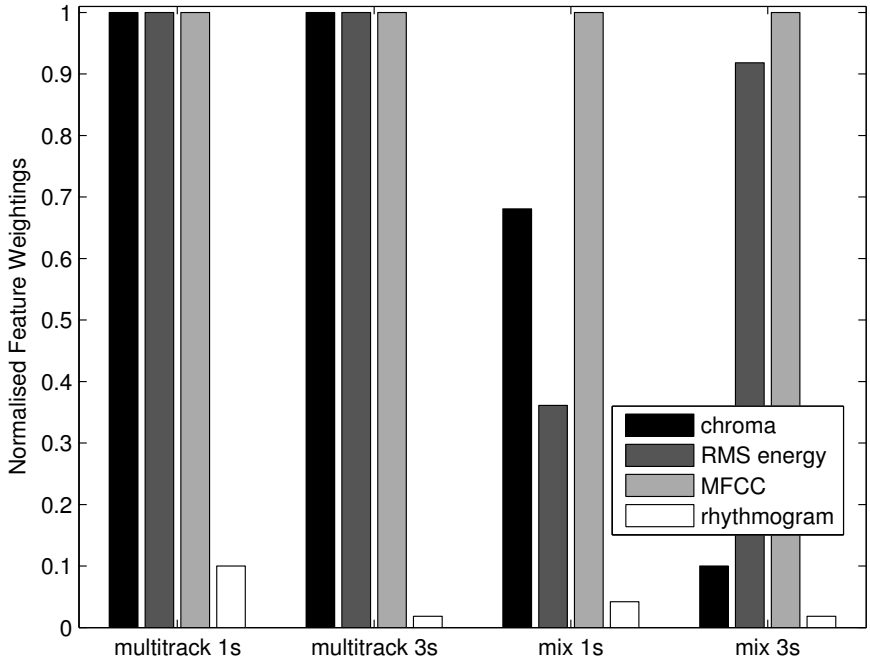

Figure 4.7: Optimum feature weights for multitrack and mixed audio sources, using 1 second and 3 second tolerances

exactly equal. Additionally, when using Brennan's peak-picking algorithm, the second, or mid, level of segmentation produced was found universally to be optimum. In the cases where we used the simple peak-picking method, the optimum peak height threshold ranged from 1 to 1.5 times the average peak height.

4.4.4 Discussion

The most important results in Table 4.1 demonstrate the significant improvement in F-value, precision and recall achieved when we make full use of the multitrack data as opposed to just the final mix (the F-value rises from 0.30 for full mix data at a 1s tolerance, to 0.38 for multitrack data when we use Brennan's peak-picking algorithm, and likewise from 0.53 to 0.6 at a 3s tolerance). This result strongly supports our hypothesis; that having access to multitrack data enables an increase in segmentation accuracy.

A slightly surprising result is that Mauch's algorithm only achieves similar

or worse F-values to our own, relatively simple, algorithm when it too is applied to the full mixes. It is worth noting though that Mauch's algorithm has both the highest precision and the lowest recall values of all the methods, indicating that although a lot of true segment boundaries were missed altogether, those that were produced were relatively accurate compared when to the other algorithms. This, together with the fact that our algorithm achieved the best results when we used the second (mid) level of segmentation, perhaps implies that typically there are higher numbers of segment boundaries present in our ground truth data than in that used for the MIREX 2009 segmentation task. Indeed, closer inspection of both sets of ground truths reveals that, on average, each MIREX ground truth song contains 11.2 segment boundaries whilst our SALAMI style ground truths contain 30. The MIREX ground truth data was not, as far as we can ascertain, produced according to the SALAMI guidelines. This goes some way to explain the lower F-values achieved using Mauch's algorithm, whilst not invalidating the improvement observed when applying our own algorithm to multitrack rather than mixed audio.

Our algorithm produced optimum results when analysing full multitrack data by using equal weightings of chroma, RMS energy and MFCC features (compared to the dominance of MFCCs when analysing fully mixed songs); this result demonstrates that given a large collection of recordings of different instruments, no one particular feature stands out as universally appropriate. Accordingly we investigate this result further in Section 4.5. Additionally, the particular method of segment boundary picking used here is designed to search for regions of homogeneity, however repetition is also an important cue in structural segmentation. A further refinement of the experiment would be to incorporate a repetition-based method; several possibilities are listed by Paulus et al. (2010).

It was not entirely surprising that the rhythmogram feature scored so poorly in the optimum feature weight configurations. This feature is calculated over a relatively large time window (2s), resulting in poor temporal accuracy, whereas all other features are calculated on a frame-by-frame basis.

As an aside, an interesting observation was made during some early tests when our dataset was much smaller (approximately 20 songs). One song, the nature of which happened to be a contemporary, relatively experimental piece based mainly upon the presence or absence of subtle layers of instruments and

loops, had to be taken out of the test dataset for copyright reasons. It was replaced by a far more traditional pop/rock song from the late sixties, and the effect was a degradation of the F-values achieved using the multitrack data, and concurrently an improvement in those obtained from the final mixes. Whilst this is certainly not a robust result, it is interesting in that it suggests that certain genres of music which don't follow the traditional verse/chorus pattern are more easily segmented when we have access to the multitrack recordings, in which subtle changes in instrumentation are more amenable to analysis.

4.5 Instrument-Specific Audio Features

The experiment described in Sections 4.4.1 to 4.4.3 results in more accurate segmentations of our test dataset by analysing equally weighted chroma, RMS energy and MFCC features from multitrack rather than mixed audio data (the optimum weighting of the rhythmogram audio feature was found to be as low as possible – i.e., it was not beneficial to use it). It achieves this whilst paying no attention to the specific nature of each source audio track – that is, all tracks are treated identically, regardless of musical instrument type. When alternative feature weights are tested, they are applied uniformly to all audio tracks. Intuitively though, one might expect certain types of audio feature to be more appropriate for certain types of audio track. Chroma features, for example, are theoretically more applicable to detecting changes in the melodic or harmonic properties of an audio track, whereas MFCC features are better suited to identifying the timbrel qualities of audio. We might therefore expect to be able to improve our segmentation results further by using specific audio features for source tracks containing specific types of musical instrument – consequently we perform an additional experiment to test this hypothesis.

4.5.1 Feature Selection

We need to decide which audio feature to analyse for each audio track, but first, a pre-requisite to feature selection is the ability to classify our source audio tracks as belonging to one of a discrete number of specific musical instrument types. Of the 936 source audio tracks in our dataset, 859 (92%) have filenames easily recognisable as common instrument names. The remaining 77 tracks (8%) have uninformative or ambiguous names, such as 'track1', or 'Kanonaki

RODE-04'. Using heuristics, we extract the following main instrument types from our collection of audio source track filenames:

- Brass
- Keys
- Percussion
- Plucked strings
- Bowed strings
- Vocal
- Woodwind
- Fx
- Bass

Figure 4.8 shows a histogram of the numbers of audio tracks falling into each category (with the additional category 'unclassified' for the 77 ambiguously named files), whilst Table 4.2 shows the mappings between instrument category and source audio track filename keywords.

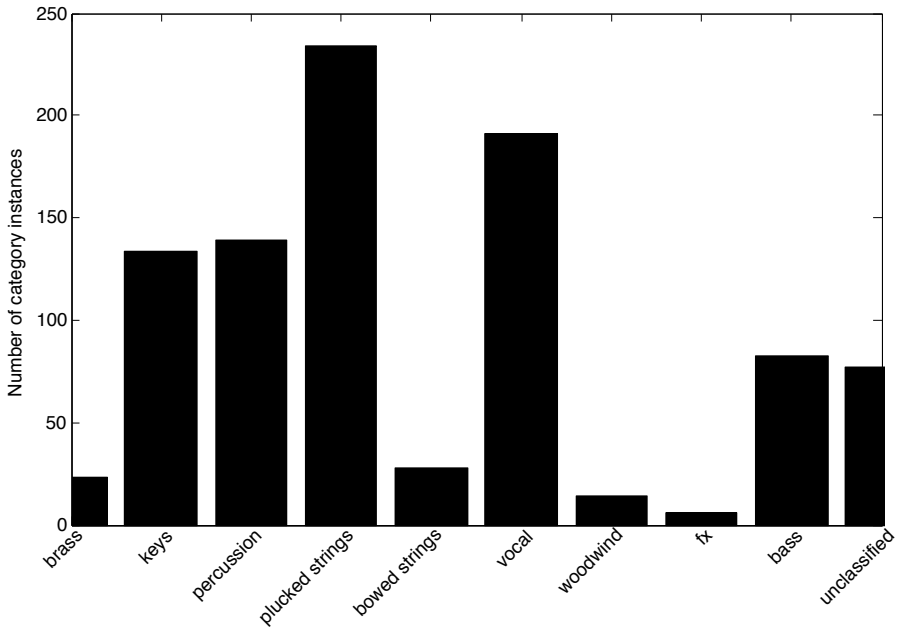

Figure 4.8: Number of source audio tracks per instrument category

In order to determine which audio feature to use for each instrument category, we could take a similar approach to the one taken in our previous experiment – for example we could run repeat experiments for every possible configuration of audio feature / instrument category mappings. The amount of computations involved though for a dataset of this size make this approach impractical, and so instead we select our own audio feature / instrument category mapping according to our intuitions regarding the nature of each musical instrument category. To illustrate, Figures 4.9 and 4.10 show the chroma and MFCC audio feature spectrograms of, respectively, a solo oboe recording and a solo electric, over-driven, rhythm guitar recording. At the top of both figures we have manually annotated what appear to be distinct segments within the songs – the letters A, B, C and D represent distinct segment types.

Instrument Category	**Filename Keywords**
Brass	brass, horn, saxophone, trombone, trumpet, tuba
Keys	piano, pad, keyboard, synth, korg, rhodes, key, organ, wurlitzer, casio
Percussion	drum, beat, bell, bongo, conga, cowbell, glockenspiel, clap, vibes, roll, kick, snare, percussion, bang, hat, tamb, shaker
Plucked strings	guitar, rhythm, wah, banjo, koto, sitar, pizzicato, gtr, harp, mandolin, harpsichord, nylon, acousticg, electricg
Bowed strings	strings, cello, violin, viola
Vocal	vocal, accapela, accapella, speech, voice, vox
Woodwind	bassoon, clarinet, flute, oboe, piccolo, picollo, wind
Fx	effect
Bass	bass

Table 4.2: Audio source track filename keyword to musical instrument category mappings

In the case of the oboe (Figure 4.9), both the chroma and MFCC features exhibit clear block structure. Of the two, chroma seems to perform particularly well, with a very clear distinction between prominent notes, displayed in red, and the unsounded notes, displayed in green Although the same repeat structure is visible for the MFCC features, the contrasts both within and between segments, at least to the human eye, are less pronounced. In Figure 4.10, the chroma features spectrogram shows far less contrast between note values than the corresponding image for oboe. This is possibly due the the large number of harmonic overtones typical of an overdriven electric guitar. In contrast, the MFCC features spectrogram looks almost entirely uniform during the non-silent portions of the song.

We speculate that chroma audio features are most suitable for musical instruments exhibiting clear fundamental frequencies, MFCC audio features to instruments having more complex harmonics, and RMS energy to more transient instruments with high dynamic range and broad frequency spectrum. Where audio tracks have ambiguous filenames, we default, given its prevalence in previous segmentation algorithms (Paulus and Klapuri, 2008; Bruderer et al., 2006; Aucouturier et al., 2005), to the MFCC feature. Consequently we select audio

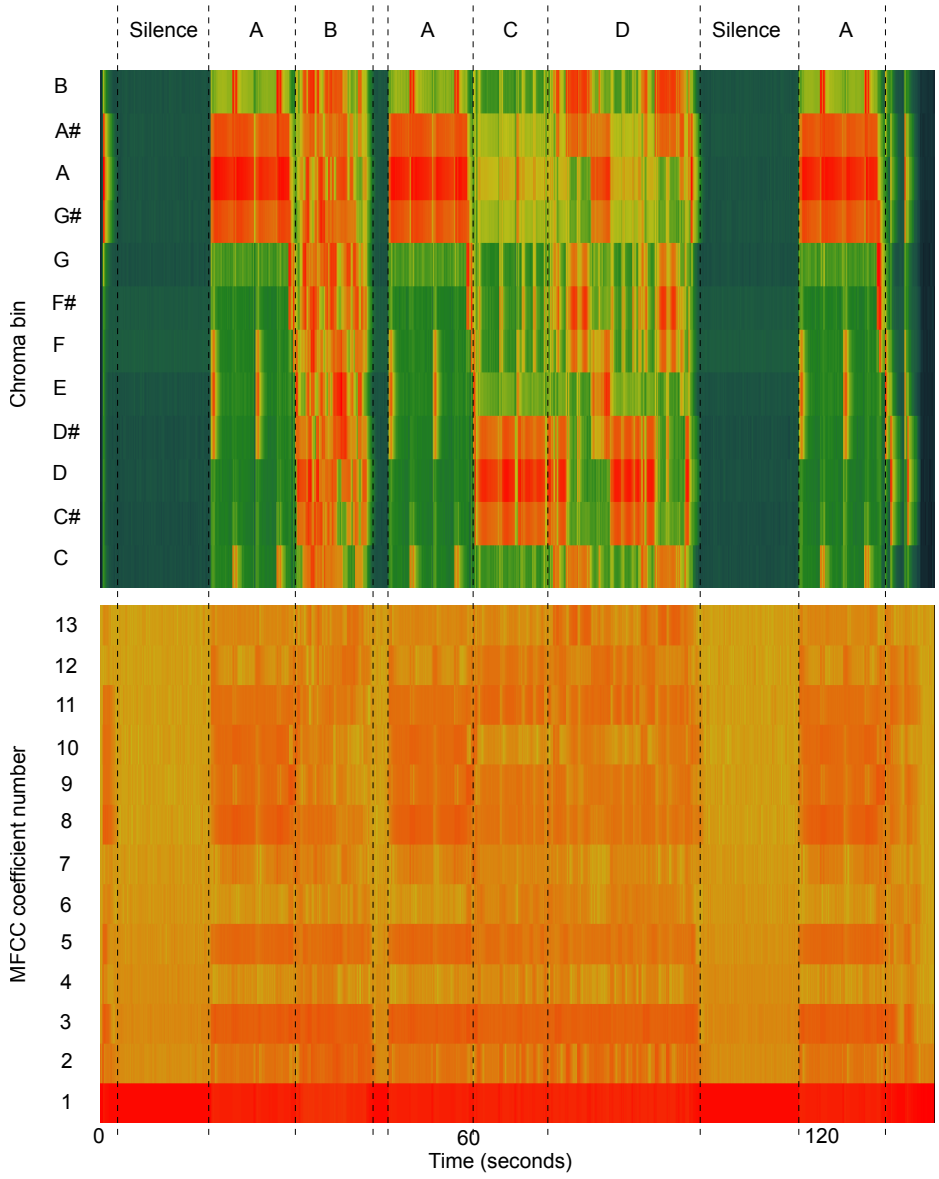

Figure 4.9: Chroma and MFCC audio feature spectrograms for oboe

features for each instrument category as shown in Table 4.3.

4.5.2 Experimental Method

Our experimental method is the same as for the segmentation experiment described in Section 4.4.1 of Chapter 4, except for two differences – we only calculate one type of audio feature per track (determined using the track name

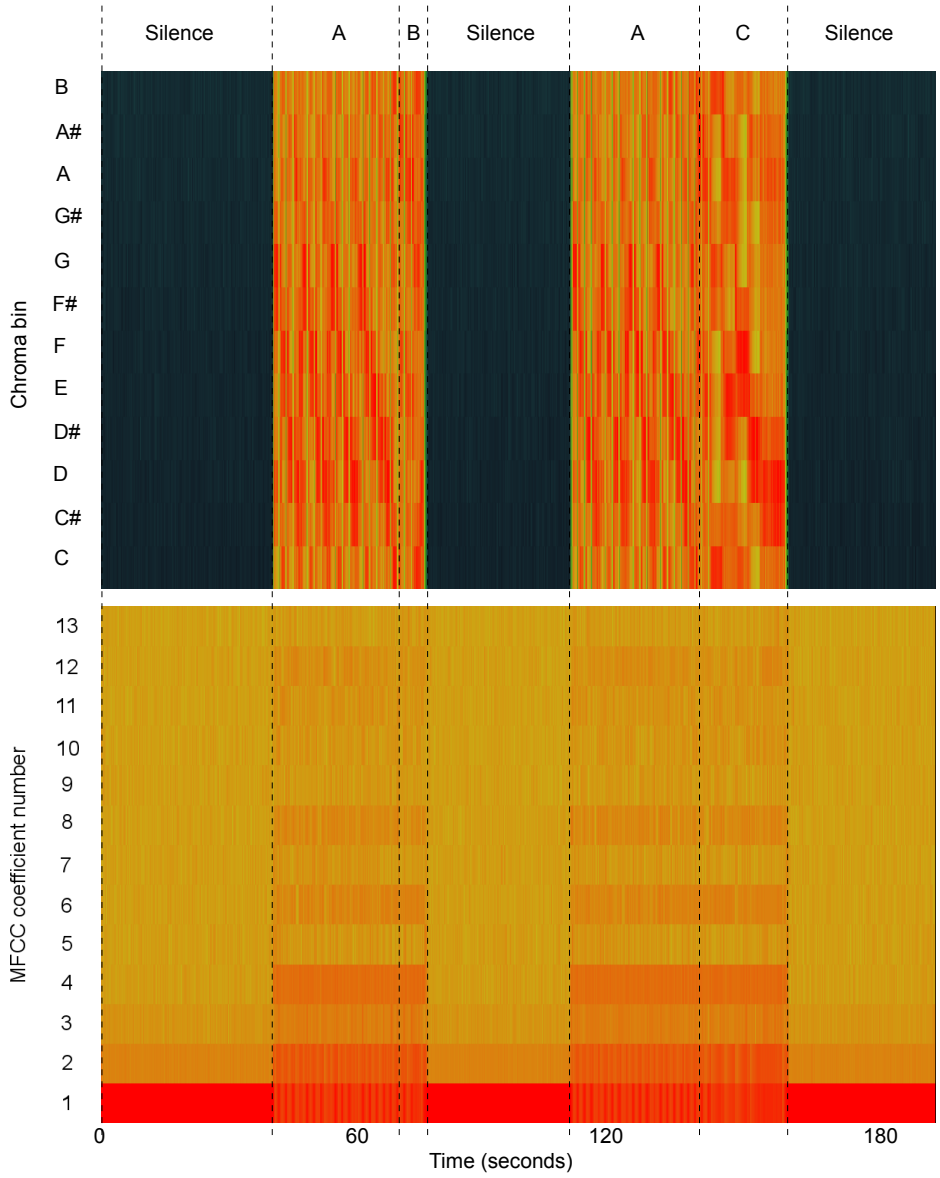

Figure 4.10: Chroma and MFCC audio feature spectrograms for electric overdriven rhythm guitar

to feature mappings of Table 4.3), and only calculate self-distance matrices (and therefore also segment boundary locations) using equally weighted features, rather than attempting to find an optimum feature weight set. We then calculate self-distance matrices, novelty curves and finally segment boundary locations as before.

Instrument Category	Audio Feature
Brass	MFCC
Keys	chroma
Percussion	RMS energy
Plucked strings	MFCC
Bowed strings	chroma
Vocal	MFCC
Woodwind	chroma
Fx	MFCC
Bass	chroma
Unclassified	MFCC

Table 4.3: Musical instrument category to audio feature type mappings

4.5.3 Evaluation

Our method of evaluation is exactly the same as that described in Section 4.4.2.

4.5.4 Results

Table 4.4 shows the results (on the bottom row) of this new experiment when we use a 1s tolerance for matching segment boundaries, along with the results we obtained previously in Section 4.4.3 for comparison.

Experimental Method	1 second tolerance		
	F-value	Precision	Recall
Mauch, applied to mixdowns	0.29	0.44	0.22
Hargreaves, applied to mixdowns (Brennan's peak pick)	0.30	0.29	0.31
Hargreaves, applied to multitracks (Brennan's peak pick)	0.38	0.37	0.39
Hargreaves (instrument-specific features and Brennan's peak pick), applied to multitracks	**0.32**	0.31	0.33

Table 4.4: Segment boundary retrieval comparisons with ground truth data, using instrument-specific features

Our new F-value, 0.32 (highlighted in bold), is a slight improvement compared to either of the previous techniques applied to mixed audio. However it is inferior to the result obtained previously from multitrack data (0.38), when all tracks were treated identically.

4.5.5 Discussion

This particular configuration of instrument categories and instrument category to audio feature mappings does not improve upon the results from the combined and weighted audio features experiment of Section 4.4. This fact though does not preclude the possibility that there may be other instrument grouping and feature mapping permutations which would perform better. However the levels of computing resources necessary to find such a permutation renders an optimisation experiment impractical. To illustrate, some example execution times and results file sizes for specific computer hardware are given below:

- Hardware Specification: 12 x 2GHz CPU cores (with two threads/core hyperthreading), 128GB of memory, 64-bit Linux
- Full combined and weighted features experiment (as described in Section 4.4) execution time: 5.5 days
- Example results datafile size (audio features and self-distance matrices for one permutation of audio features and 104 songs): 6 GB
- Calculating the self-distance matrices for just *one* permutation of instrument category to audio feature mappings (as necessary for the instrument-specific audio features experiment), for all 104 songs, execution time: 30 minutes

We have nine instrument categories and three audio feature types, which gives us $3^9 = 19683$ permutations of instrument category to audio feature mappings. To calculate the self-distance matrices for all those permutations would take approximately 410 days. We could however assign a feature type to each instrument independently of the others and adopt a hill climbing search strategy – this would reduce the humber of permutations to $1 + (3 - 1) * 9 = 19$, giving an execution time of around 9.5 hours. Additionally though we would also need to experiment with different instrument groupings, increasing execution time into days rather than hours.

4.6 Conclusion

Many methods exist for determining the high-level structure of fully mixed musical audio. Inevitably, all of these methods need to extract relevant musical

cues from the ensemble of instruments present in most recordings. We have shown that there is a quantifiable and significant advantage to be gained, when segmenting music, by exploiting the source-separate versions of audio recordings if they are available as multitrack projects. We have used a relatively simple algorithm based upon four audio features, self-distance matrices and homogeneity detection, which pays no attention to the particular type of instrument present in each source track, and we predict that even greater segmentation accuracy and/or reduced computational complexity could be achieved by selecting audio features according to the instrumentation or musical function of each track.

It has been implicitly assumed so far that at the point of analysis, all the source audio tracks required prior to producing the final mixdown are present. However, at intermediate stages of the recording process, only a subset of these tracks will exist. An interesting direction for future work would therefore be to determine how accurately we are able to segment incomplete subsets of multitrack projects. Answering this question will also enable us to establish whether or not either a single or a minimal number of tracks of certain instrument combinations are sufficient for the derivation of an accurate segmentation, without needing to perform analysis of tracks which offer little or no new information. Given that DAW multitrack projects frequently contain around 8, 16 or 24 tracks, this would have the added benefit of greatly reducing the computational complexity of our segmentation algorithm.

In addition to high-level verse/chorus type segmentations, we should also expect to be able to achieve lower (i.e. bar and beat) level segmentations. Possible ways to achieve this might be by analysis of the sub-structure of self-distance matrices (i.e. recursively analyse a self-distance matrix after first identifying high-level boundaries) or by using existing beat tracking or transcription algorithms, for example.

In this chapter we have only discussed methods of locating segment boundaries, however we do not have to limit ourselves to this narrow goal. To date, and to the authors' knowledge, all MIR research – be that structural segmentation, genre/artist/mood classification, music similarity measurement, onset/key detection, cover song identification, or chord/melody extraction, has been undertaken using either fully mixed music or single instrument recordings. The potential advantages offered by early capture of a more accurate and rich set of metadata from multitrack sources in the studio are vast, and, because the

metadata need not stay tightly bound to the commercial audio recording, are not limited to the improvement of studio editing tools. Technologies such as the World Wide Web Consortium (W3C) RDF metadata model are already being used to enhance on-line artist information websites such as that provided by the BBC[14], and the publication of enhanced metadata alongside commercial audio recordings would only increase their versatility. Instead of concentrating on complex signal processing forms of 'reverse engineering' fully mixed audio, the MIR community may instead concentrate on exploiting an already present, easy to query and potentially vast amount of metadata via logical inferencing. On this basis, the next chapter is devoted to exploring the viability of performing the type of algorithmic data analysis commonly encountered within symbolic music data MIR, using multidimensional symbolic music data expressed in RDF.

[14] `http://www.bbc.co.uk/music`

Chapter 5

A Semantic Web Approach to Pattern Discovery in Data and Music

In the previous chapter we described some experiments in which we exploited the multitrack versions of music recordings in order to achieve significantly more accurate structural segmentations of the songs, compared to the more typical case of using the final mixed recordings. Together with the discussion of single versus multi-instrument MIR we provided in Section 2.1 of Chapter 2, we speculate that similar improvements in accuracy could be achieved by performing other MIR tasks similarly early in the music recording and production process.

Although of course we are still a considerable distance from being able to generate a high-accuracy, MIDI-like score from audio recordings, it seems appropriate to look ahead now and start to investigate not only what kind of musicological insights we might derive from such a symbolic score, but also, how we might best share this type of metadata in order that the MIR community or, for example, the wider music industry, may extract maximum benefit from it.

We have seen in Section 2.3.4 that the Semantic Web is being widely espoused as a beneficial way to share and make sense of data on the internet, and we briefly introduced the SIA and SIATEC symbolic data pattern discovery algorithms in Section 2.4.2. We know that it is possible to *link* (Section 2.3.1) and query (Section 2.3.2) RDF data, and to create Semantic Web ontologies in order to make the meaning of our data explicit (Section 2.3.3). What is less clear though is whether or not it is either possible or practical to make use of Semantic Web technologies in order to perform the type of music analysis

usually carried out by more conventional programming languages.

Accordingly, in this chapter, we continue our exploration of the vision set out in Chapter 3 by describing in detail a new, Semantic Web implementation of the SIA and SIATEC algorithms (Meredith et al., 2002), alongside mathematical definitions of the algorithms' functions. We present important performance evaluation metrics, which, to the author's knowledge, are unique within the field of MIR in shedding light upon the viability of current Semantic Web technology implementations applied to this type of content-analysis task.

There is a great deal of interest in, and a general drive towards, using the Semantic Web as a means of sharing machine-readable music metadata in a non-proprietary format. The far-reaching Online Music Recognition and Searching (OMRAS 1 and 2) projects,[1] for example, set out to promote and enable such activities, and resulted in the development of the Music (Raimond et al., 2007), Audio Features[2], VAMP Plugin (Cannam et al., 2010), and Chord[3] ontologies, as well as 103 related research publications. As take up of Semantic Web data publishing increases, we find ourselves with an ever increasing amount of publicly accessible linked data, and ever more ontologies describing the data relationships. The question remains, though, what else can we do with such an abundance of metadata?

For example, the audio features ontology allows us to describe low-level, signal-processing-derived audio data (frame-level note onsets or Mel Frequency Cepstral Coefficients, for instance) using RDF. This raw data though, on its own, is of little value – inevitably we will want to perform further data processing in order to find whatever it is we're looking for (identification of the musical instrument used to play the piece, or whether or not the original piece was in a twelve bar blues form, for example). At this stage the next step is usually to break out of the Semantic Web domain and back into some conventional programming language, which unfortunately means our processing algorithm and data format is not necessarily any longer in a machine-readable, programming language-independent form, thus, to some extent, we've defeated the object of using the Semantic Web in the first place. Alternatively, it is also common to perform extensive signal or data processing in a more conventional programming

[1] `http://omras2.org/`

[2] `http://motools.sourceforge.net/doc/audio_features.html`

[3] `http://purl.org/ontology/chord/`

language first, and then to publish the resulting high-level metadata as RDF. Whilst it is possible to provide sufficient metadata about an algorithm and its parameters such that all aspects of the processing workflow remain open and accessible (the VAMP Plugin ontology, for example, goes some way to achieving this by providing a mechanism for stating the output feature type of a VAMP audio plugin[4] – see Cannam et al., 2010), and even to trigger an external signal processing computation from SPARQL (for example, Henry[5] is an attempt to implement a "DSP-driving SPARQL end point, based on transaction logic"), we still face the inevitable complications of combining multiple technologies in order to achieve our goals. To date (at least not to the authors' knowledge), Semantic Web technologies have rarely been used to actually perform any kind of algorithmic content analysis of the music itself – one exception being the use of a 'Harmony' ontology (Ibbotson, 2009). This ontology is intended to provide a "harmonic structural model of music", and, accordingly, contains ontological definitions of notes, chords and keys. In terms of inferencing capabilities though, it simply defines a `chordFollows` predicate as the `owl:inverseOf` a `chordPrecedes` predicate, thus allowing us to infer (for example) that chord 'b' follows chord 'a' if we had previously asserted that chord 'a' precedes chord 'b'. We would like to perform significantly more complex content analysis than this.

The SIA family of pattern discovery algorithms (Meredith et al., 2002) are examples of commonly-encountered, 3SUM-hard (Clifford et al., 2006), cross-product type algorithms (other examples include the many audio-based music structure analysis algorithms dependant upon self-distance matrix calculations, first described by Foote, 2000). Given the commonality of such algorithms within MIR, and the growing interest in the use of Semantic Web technologies as practical knowledge engineering tools, we present here an investigation into the viability of applying Semantic Web technologies (RDF, SPARQL 1.1, and OWL 2) to the task of algorithmic content analysis (specifically, pattern discovery in symbolic data). We address the question, therefore, of whether or not it is possible to replicate the kind of programmatic functionality and flexibility offered to us by more conventional programming languages, with only

[4] `http://vamp-plugins.org`

[5] `http://code.google.com/p/km-rdf/`

RDF, SPARQL and OWL (together with an appropriate OWL reasoner) as our toolkit. We describe the challenges presented by using this approach, the particular solution we arrived at, and an evaluation of the performance of our method when compared to a purely Java implementation of the same algorithm.

5.1 The SIA and SIATEC Algorithms

5.1.1 Overview

The Structure Induction Algorithm (SIA) and Structure Induction Algorithm Translational Equivalence Class (SIATEC) algorithms are pattern discovery algorithms acting upon symbolic data. Fundamental to both algorithms are *Datapoints* and *Vectors*. SIA and SIATEC datapoints are k-dimensional points in Euclidean space, whose coordinates are the values of each of the attributes of a single musical artefact (most commonly a musical note or beat) which we have assigned to that particular dimension. A simple two-dimensional example would be note onset time in one dimension and chromatic pitch in the other. A full musical score, such as the one shown in Figure 5.1, would then be represented by a collection of these datapoints, as illustrated in Figure 5.2.

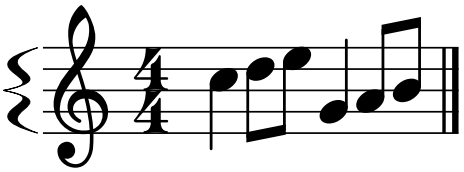

Figure 5.1: Simple score

The purpose of the SIA algorithm is to find all of the distinct collections of m datapoints, referred to as Maximal Translatable Patterns (MTPs), each of which may be translated by some k-dimensional vector $\boldsymbol{v}$ onto another set of m datapoints within the same dataset, where m is the largest number of datapoints which may be translated by $\boldsymbol{v}$. This identifies all of the (theoretically) most perceptually significant patterns within our dataset, but does not show all the locations of all of the repeats. The SIATEC algorithm does precisely that – identifying the collection of vectors by which each MTP may be mapped onto other datapoints within the same dataset. After removing duplicates (e.g., if

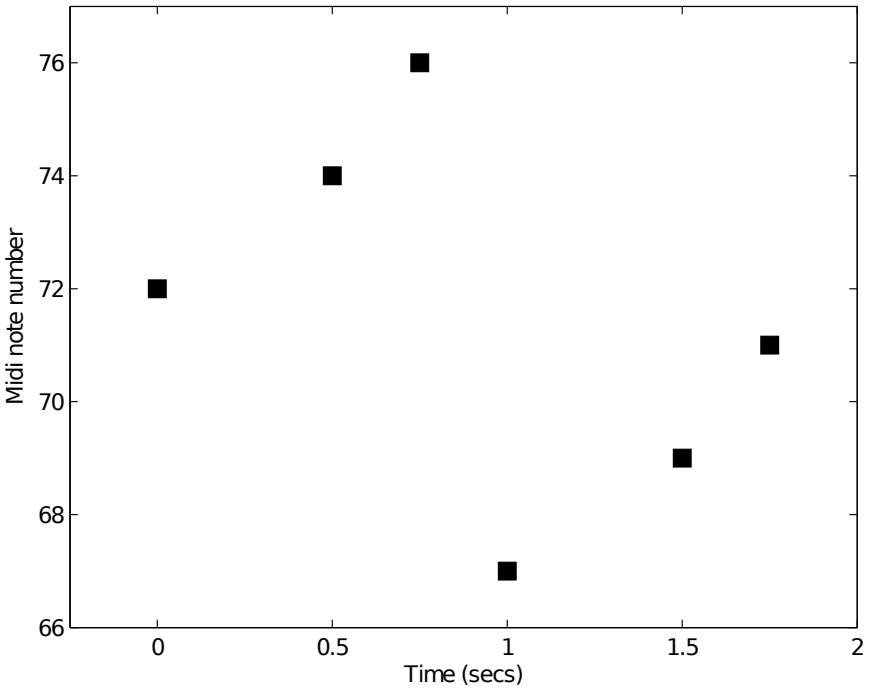

Figure 5.2: Simple score midi note onsets

we can move MTP A to MTP B by the vector $\langle 2, 1\rangle$, then we can also move MTP B to MTP A by the vector $\langle -2, -1\rangle$ – we only need record one of this pair of results), we refer to each MTP and its associated collection of vectors as a Translational Equivalence Class (TEC). Figure 5.3 shows that the pair of datapoints $\langle 0.5, 74\rangle, \langle 1.5, 69\rangle$ from Figure 5.2 can be translated onto other datapoints in our set by either of the two vectors $\langle 0.25, 2\rangle$, and $\langle -0.5, -2\rangle$, hence we have the TEC:

$$< \{\langle 0.5, 74\rangle, \langle 1.5, 69\rangle\}, \{\langle 0.25, 2\rangle, \langle -0.5, -2\rangle\} >$$

Figure 5.4 shows another example TEC from the same score; this time a collection of three datapoints can be translated by just one vector.

Our aim in this chapter is not to evaluate or discuss the suitability of the SIA and SIATEC algorithms for discovering patterns in music. For that, many SIATEC-based algorithm evaluations and comparisons exist – see for example Clifford et al. (2006); Collins et al. (2010); Collins (2011); Collins and Meredith (2013). Instead, we focus purely on the viability of implementing such an

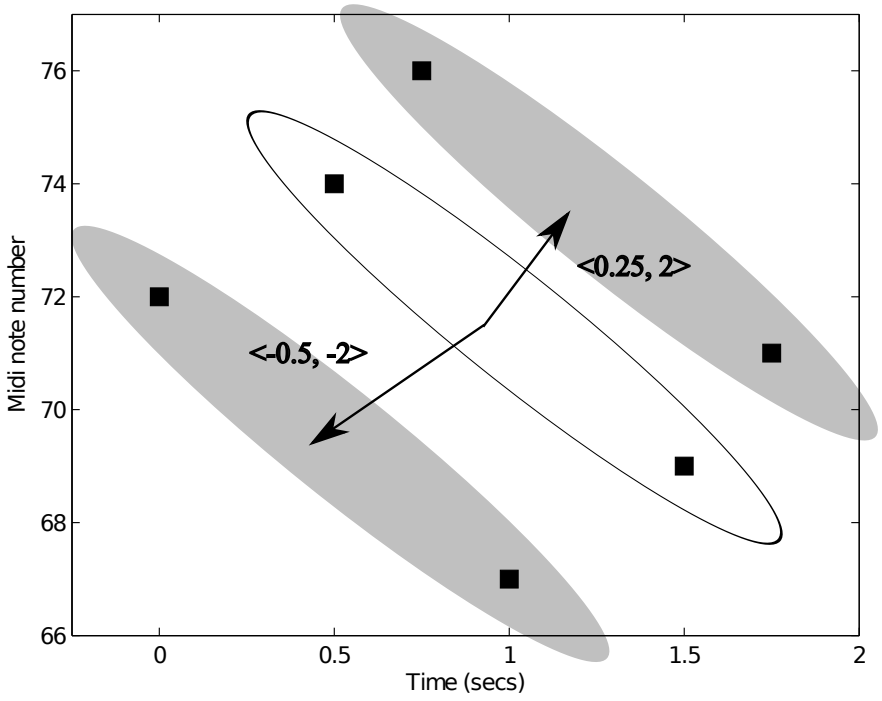

Figure 5.3: Simple score with TEC A

algorithm using Semantic Web technologies, and evaluating how well the implementation at which we have arrived performs compared to a non-Semantic Web version.

5.1.2 Algorithm Definitions

The complete mathematical definitions of the functions computed by the SIA and SIATEC algorithms, together with proofs where appropriate and specific function implementation details, are given by Meredith et al. (2002). For clarity we provide mathematical definitions here too, although we omit the proofs and the particular function implementations specific to that paper.

Datapoints Dataset

Our musical score will have a finite number of notes, each of which can be represented by a k-dimensional vector. We therefore have a dataset D containing n datapoints, and we denote a single datapoint by the vector $\boldsymbol{d}$. For our example

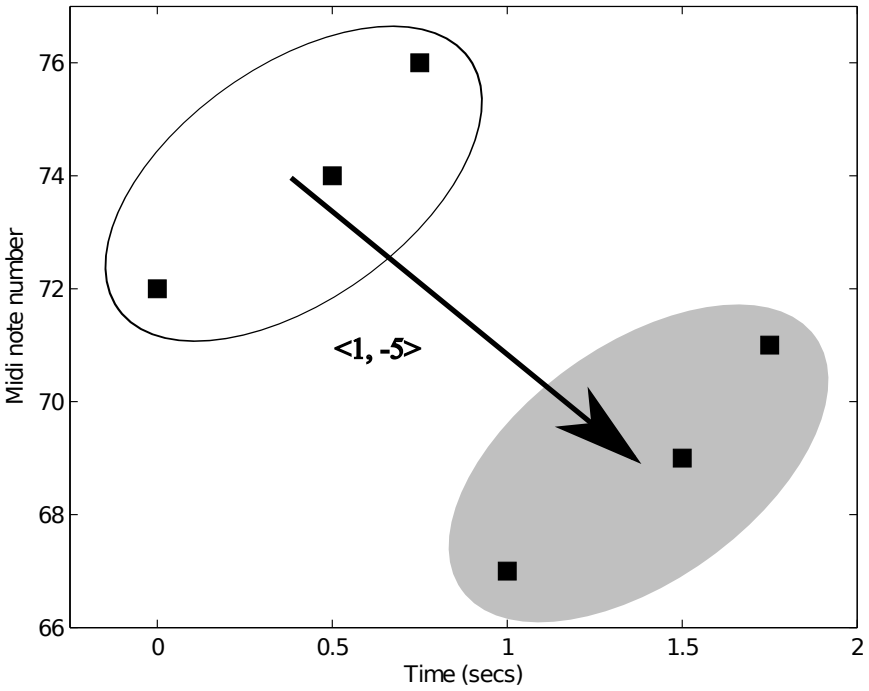

Figure 5.4: Simple score with TEC B

MIDI score shown in Figure 5.2, we have $k = 2$ dimensions, with note onset time in dimension 1, and MIDI pitch in dimension 2. In no particular order, the datapoints are represented by the following vectors:

$\langle 1, 67 \rangle$

$\langle 1.5, 69 \rangle$

$\langle 1.75, 71 \rangle$

$\langle 0, 72 \rangle$

$\langle 0.5, 74 \rangle$

$\langle 0.75, 76 \rangle$

Vectors

Subtracting any datapoint $\boldsymbol{d_1}$ from any other datapoint $\boldsymbol{d_2}$ gives us the vector $\boldsymbol{v}$

$$\boldsymbol{v} = \boldsymbol{d_2} - \boldsymbol{d_1} \tag{5.1}$$

which also implies,

$$\boldsymbol{d_2} = \boldsymbol{v} + \boldsymbol{d_1} \tag{5.2}$$

This means that the datapoint $\boldsymbol{d_1}$ is *translatable* onto $\boldsymbol{d_2}$ by the vector $\boldsymbol{v}$. Using one pair of datapoints from our example dataset, we can say that the vector $\langle 1, -5 \rangle$ translates the datapoint $\langle 0.5, 74 \rangle$ onto the datapoint $\langle 1.5, 69 \rangle$.

Vector Order Relation

We denote the ith element of vector $\boldsymbol{a}$ by $\boldsymbol{a}[i]$. For two k-dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, where $k \geq 1$, we define the *less than* relationship as

$$\begin{aligned} \boldsymbol{u} < \boldsymbol{v} \iff & (\boldsymbol{u}[1] < \boldsymbol{v}[1]) \\ & \vee \\ & (\exists i : (1 < i \leq k) \wedge (\boldsymbol{u}[i] < \boldsymbol{v}[i]) \wedge (\forall j \in \{1 \leq j < i\} : \boldsymbol{u}[j] = \boldsymbol{v}[j])) \end{aligned} \tag{5.3}$$

In simpler terms, starting at the lowest dimension ($i = 1$), compare the values of $\boldsymbol{u}[i]$ and $\boldsymbol{v}[i]$. If $\boldsymbol{u}[i] < \boldsymbol{v}[i]$, then $\boldsymbol{u} < \boldsymbol{v}$. If however $\boldsymbol{u}[i] = \boldsymbol{v}[i]$, then increment the value of i and compare values again. If the values are found to be equal in all dimensions, then $\boldsymbol{u} \geq \boldsymbol{v}$ (in fact, in that case, although it is not made explicit in Equation 5.3, we can say $\boldsymbol{u} = \boldsymbol{v}$). So for example, $\langle 3, 2 \rangle < \langle 3, 3 \rangle < \langle 4, 1 \rangle$. Determining the ordered set $\boldsymbol{D}$ of datapoints in our dataset D is a necessary step of both the SIA and SIATEC algorithms, and for our example dataset, the ordered datapoints together with their ordered indices are shown in Table 5.1.

Datapoint	**Ordered Index**
$\langle 0, 72 \rangle$	1
$\langle 0.5, 74 \rangle$	2
$\langle 0.75, 76 \rangle$	3
$\langle 1, 67 \rangle$	4
$\langle 1.5, 69 \rangle$	5
$\langle 1.75, 71 \rangle$	6

Table 5.1: Set $\boldsymbol{D}$, the ordered set of datapoints

Vector Tables

For both the SIA and the SIATEC algorithms, using the notation $|\boldsymbol{A}|$ to denote the cardinality of the set or vector $\boldsymbol{A}$, we calculate the set

$$V = \{\langle \boldsymbol{D}[j] - \boldsymbol{D}[i], i\rangle | 1 \leq i < j \leq |\boldsymbol{D}|\} \tag{5.4}$$

that is, the set of ordered pairs consisting of all the vectors $\boldsymbol{v}$ between all pairs of datapoints $\boldsymbol{d_1}$ and $\boldsymbol{d_2}$, as well as the ordered index of the originating datapoint $\boldsymbol{d_1}$, subject to the condition that the ordered index of $\boldsymbol{d_1}$ is less than the ordered index of $\boldsymbol{d_2}$. We record these vectors in a *vector table*, as shown in Table 5.2 for our example dataset. We call this vector table V.

		From Datapoint					
		$\langle 0, 72\rangle$	$\langle 0.5, 74\rangle$	$\langle 0.75, 76\rangle$	$\langle 1, 67\rangle$	$\langle 1.5, 69\rangle$	$\langle 1.75, 71\rangle$
To Datapoint	$\langle 0, 72\rangle$						
	$\langle 0.5, 74\rangle$	$\langle\langle 0.5, 2\rangle, 1\rangle$					
	$\langle 0.75, 76\rangle$	$\langle\langle 0.75, 4\rangle, 1\rangle$	$\langle\langle 0.25, 2\rangle, 2\rangle$				
	$\langle 1, 67\rangle$	$\langle\langle 1, -5\rangle, 1\rangle$	$\langle\langle 0.5, -7\rangle, 2\rangle$	$\langle\langle 0.25, -9\rangle, 3\rangle$			
	$\langle 1.5, 69\rangle$	$\langle\langle 1.5, -3\rangle, 1\rangle$	$\langle\langle 1, -5\rangle, 2\rangle$	$\langle\langle 0.75, -7\rangle, 3\rangle$	$\langle\langle 0.5, 2\rangle, 4\rangle$		
	$\langle 1.75, 71\rangle$	$\langle\langle 1.75, -1\rangle, 1\rangle$	$\langle\langle 1.25, -3\rangle, 2\rangle$	$\langle\langle 1, -5\rangle, 3\rangle$	$\langle\langle 0.75, 4\rangle, 4\rangle$	$\langle\langle 0.25, 2\rangle, 5\rangle$	

Table 5.2: Vector table V

Additionally, for the SIATEC algorithm only, we calculate the set

$$W = \{\langle \boldsymbol{D}[j] - \boldsymbol{D}[i], i\rangle | (1 \leq i \leq |\boldsymbol{D}|) \wedge (1 \leq j \leq |\boldsymbol{D}|)\} \tag{5.5}$$

recording the results in vector table W, as shown in Table 5.3. Note that vector table W is simply vector table V, but with the 'missing' elements now present.

		From Datapoint					
		$\langle 0, 72\rangle$	$\langle 0.5, 74\rangle$	$\langle 0.75, 76\rangle$	$\langle 1, 67\rangle$	$\langle 1.5, 69\rangle$	$\langle 1.75, 71\rangle$
To Datapoint	$\langle 0, 72\rangle$	$\langle\langle 0, 0\rangle, 1\rangle$	$\langle\langle -0.5, -2\rangle, 2\rangle$	$\langle\langle -0.75, -4\rangle, 3\rangle$	$\langle\langle -1, 5\rangle, 4\rangle$	$\langle\langle -1.5, 3\rangle, 5\rangle$	$\langle\langle -1.75, 1\rangle, 6\rangle$
	$\langle 0.5, 74\rangle$	$\langle\langle 0.5, 2\rangle, 1\rangle$	$\langle\langle 0, 0\rangle, 2\rangle$	$\langle\langle -0.25, -2\rangle, 3\rangle$	$\langle\langle -0.5, 7\rangle, 4\rangle$	$\langle\langle -1, 5\rangle, 5\rangle$	$\langle\langle -1.25, 3\rangle, 6\rangle$
	$\langle 0.75, 76\rangle$	$\langle\langle 0.75, 4\rangle, 1\rangle$	$\langle\langle 0.25, 2\rangle, 2\rangle$	$\langle\langle 0, 0\rangle, 3\rangle$	$\langle\langle -0.25, 9\rangle, 4\rangle$	$\langle\langle -0.75, 7\rangle, 5\rangle$	$\langle\langle -1, 5\rangle, 6\rangle$
	$\langle 1, 67\rangle$	$\langle\langle 1, -5\rangle, 1\rangle$	$\langle\langle 0.5, -7\rangle, 2\rangle$	$\langle\langle 0.25, -9\rangle, 3\rangle$	$\langle\langle 0, 0\rangle, 4\rangle$	$\langle\langle -0.5, -2\rangle, 5\rangle$	$\langle\langle -0.75, -4\rangle, 6\rangle$
	$\langle 1.5, 69\rangle$	$\langle\langle 1.5, -3\rangle, 1\rangle$	$\langle\langle 1, -5\rangle, 2\rangle$	$\langle\langle 0.75, -7\rangle, 3\rangle$	$\langle\langle 0.5, 2\rangle, 4\rangle$	$\langle\langle 0, 0\rangle, 5\rangle$	$\langle\langle -0.25, -2\rangle, 6\rangle$
	$\langle 1.75, 71\rangle$	$\langle\langle 1.75, -1\rangle, 1\rangle$	$\langle\langle 1.25, -3\rangle, 2\rangle$	$\langle\langle 1, -5\rangle, 3\rangle$	$\langle\langle 0.75, 4\rangle, 4\rangle$	$\langle\langle 0.25, 2\rangle, 5\rangle$	$\langle\langle 0, 0\rangle, 6\rangle$

Table 5.3: Vector table W

Vector Table Element Order Relation

The elements of set V (vector table elements), may be ordered as follows. For two vector table elements $\langle \boldsymbol{u}, i\rangle$ and $\langle \boldsymbol{v}, j\rangle$, we define the *less than* relationship

as

$$\langle \boldsymbol{u}, i\rangle < \langle \boldsymbol{v}, j\rangle \iff (\boldsymbol{u} < \boldsymbol{v}) \vee (\boldsymbol{u} = \boldsymbol{v} \wedge i < j) \tag{5.6}$$

So for example, $\langle\langle 3, 2\rangle, 4\rangle < \langle\langle 3, 3\rangle, 1\rangle < \langle\langle 3, 3\rangle\, 2\rangle$. Table 5.4 shows the elements of set V in order.

Set V Element	**Ordered Index**
$\langle\langle 0.25, -9\rangle, 3\rangle$	1
$\langle\langle 0.25, 2\rangle, 2\rangle$	2
$\langle\langle 0.25, 2\rangle, 5\rangle$	3
$\langle\langle 0.5, -7\rangle, 2\rangle$	4
$\langle\langle 0.5, 2\rangle, 1\rangle$	5
$\langle\langle 0.5, 2\rangle, 4\rangle$	6
$\langle\langle 0.75, -7\rangle, 3\rangle$	7
$\langle\langle 0.75, 4\rangle, 1\rangle$	8
$\langle\langle 0.75, 4\rangle, 4\rangle$	9
$\langle\langle 1, -5\rangle, 1\rangle$	10
$\langle\langle 1, -5\rangle, 2\rangle$	11
$\langle\langle 1, -5\rangle, 3\rangle$	12
$\langle\langle 1.25, -3\rangle, 2\rangle$	13
$\langle\langle 1.5, -3\rangle, 1\rangle$	14
$\langle\langle 1.75, -1\rangle, 1\rangle$	15

Table 5.4: The ordered elements of set V

Maximal Translatable Patterns

For every vector $\boldsymbol{v}$ in a dataset D, there will be at least one pattern P (a set of datapoints), which is translatable by $\boldsymbol{v}$ onto another pattern within our dataset (this includes the trivial case where the pattern contains just a single datapoint). We define the *maximal translatable pattern* (MTP) for $\boldsymbol{v}$ as the largest translatable pattern for $\boldsymbol{v}$ (where in this case, by largest, we mean the one having the most datapoints), i.e.

$$MTP(\boldsymbol{v}, D) = \{\boldsymbol{d} | \boldsymbol{d} \in D \wedge \boldsymbol{d} + \boldsymbol{v} \in D\} \tag{5.7}$$

One such MTP from our example dataset, illustrated in Figure 5.4, is:

$$MTP(\langle 1, -5\rangle, D) = \{\langle 0, 72\rangle, \langle 0.5, 74\rangle, \langle 0.75, 76\rangle\}$$

The MTPs for a dataset can be found by grouping the elements of vector table V into those elements having the same vector $\boldsymbol{v}$, but distinct i values (remember, all vector table elements are of the form $\langle \boldsymbol{v}, i \rangle$). The datapoints having ordered indices i make up the MTP for vector $\boldsymbol{v}$. For example, referring to Table 5.4, we can see that the first entry, $\langle \langle 0.25, -9 \rangle, 3 \rangle$, is the only entry having the vector $\langle 0.25, -9 \rangle$. The single corresponding value of i is 3, and so, referring now to Table 5.1, we see that the datapoint having ordered index 3 is $\langle 0.75, 76 \rangle$. Therefore the MTP for vector $\langle 0.25, -9 \rangle$ is the single datapoint $\langle 0.75, 76 \rangle$.

Similarly, we can see from the next two entries in Table 5.4 that there are two vector table elements having the vector $\langle 0.25, 2 \rangle$. The two corresponding values of i are 2 and 5, so, referring to Table 5.1, we find the datapoints $\langle 0.5, 74 \rangle$ and $\langle 1.5, 69 \rangle$ which have ordered indices 2 and 5 respectively. Therefore, the MTP for the vector $\langle 0.25, 2 \rangle$ is the pair of datapoints $\langle 0.5, 74 \rangle$ and $\langle 1.5, 69 \rangle$. Using ordered pairs of vectors $\boldsymbol{v}$ and their corresponding MTPs (which are themselves sets of datapoints), we list the full set of ten MTPs for our example dataset:

$$
\begin{aligned}
\{ \quad & \langle\, \langle 0.25, -9 \rangle, \{ \langle 0.75, 76 \rangle \} \,\rangle, \\
& \langle\, \langle 0.25, 2 \rangle, \{ \langle 0.5, 74 \rangle, \langle 1.5, 69 \rangle \} \,\rangle, \\
& \langle\, \langle 0.5, -7 \rangle, \{ \langle 0.5, 74 \rangle \} \,\rangle, \\
& \langle\, \langle 0.5, 2 \rangle, \{ \langle 0, 72 \rangle, \langle 1, 67 \rangle \} \,\rangle, \\
& \langle\, \langle 0.75, -7 \rangle, \{ \langle 0.75, 76 \rangle \} \,\rangle, \\
& \langle\, \langle 0.75, 4 \rangle, \{ \langle 0, 72 \rangle, \langle 1, 67 \rangle \} \,\rangle, \\
& \langle\, \langle 1, -5 \rangle, \{ \langle 0, 72 \rangle, \langle 0.5, 74 \rangle, \langle 0.75, 76 \rangle \} \,\rangle, \\
& \langle\, \langle 1.25, -3 \rangle, \{ \langle 0.5, 74 \rangle \} \,\rangle, \\
& \langle\, \langle 1.5, -3 \rangle, \{ \langle 0, 72 \rangle \} \,\rangle, \\
& \langle\, \langle 1.75, -1 \rangle, \{ \langle 0, 72 \rangle \} \,\rangle \quad \}
\end{aligned}
$$

Translational Equivalence Classes

The pattern we arrive at when translating the pattern P by the vector $\boldsymbol{v}$, is denoted by $\tau(P, \boldsymbol{v})$. We denote translational equivalence between two patterns P_1 and P_2 by

$$P_1 \equiv_\tau P_2 \tag{5.8}$$

and we define translational equivalence as follows:

$$P_1 \equiv_\tau P_2 \iff \exists \boldsymbol{v} : \tau(P_1, \boldsymbol{v}) = P_2 \tag{5.9}$$

The TEC of a pattern P within dataset D is the set of patterns Q which are translationally equivalent to P and members of dataset D, i.e.:

$$TEC(P, D) = \{Q | Q \equiv_\tau P \wedge Q \subseteq D\} \tag{5.10}$$

Alternatively, a more compact way to express a TEC is as an ordered pair, where the first member of the ordered pair is a pattern P, and the second is the set of vectors $\boldsymbol{v}$ by which P is translatable within D, i.e.

$$TEC(P, D) = \langle P, \{\boldsymbol{v} | \tau(P, \boldsymbol{v}) \subseteq D\}\rangle \tag{5.11}$$

Finding the complete set of TECs for our dataset D, without redundancy (i.e. If TEC A maps onto MTP B, we don't need another TEC containing the datapoints of MTP B), is the ultimate purpose of the SIATEC algorithm. To find these TECs, we begin by grouping the MTPs of set D into sets of translationally equivalent MTPs, denoted by MEQ_l, where l is the index of each equivalent set. For our example dataset, we have:

$$\begin{aligned}
MEQ_1 = \{ \quad & \langle \langle 0.25, -9\rangle, \{\langle 0.75, 76\rangle\} \rangle, \\
& \langle \langle 0.5, -7\rangle, \{\langle 0.5, 74\rangle\} \rangle, \\
& \langle \langle 0.75, -7\rangle, \{\langle 0.75, 76\rangle\} \rangle, \\
& \langle \langle 1.25, -3\rangle, \{\langle 0.5, 74\rangle\} \rangle, \\
& \langle \langle 1.5, -3\rangle, \{\langle 0, 72\rangle\} \rangle, \\
& \langle \langle 1.75, -1\rangle, \{\langle 0, 72\rangle\} \rangle \quad \} \\
MEQ_2 = \{ \quad & \langle \langle 0.25, 2\rangle, \{\langle 0.5, 74\rangle, \langle 1.5, 69\rangle\} \rangle, \\
& \langle \langle 0.5, 2\rangle, \{\langle 0, 72\rangle, \langle 1, 67\rangle\} \rangle, \\
& \langle \langle 0.75, 4\rangle, \{\langle 0, 72\rangle, \langle 1, 67\rangle\} \rangle \quad \} \\
MEQ_3 = \{ \quad & \langle \langle 1, -5\rangle, \{\langle 0, 72\rangle, \langle 0.5, 74\rangle, \langle 0.75, 76\rangle\} \rangle \quad \}
\end{aligned}$$

Now, from each set MEQ_l, we extract only the MTP for which the translating vector has the smallest ordered index. These are the patterns P_l for each of our TECs – for our example datatset, these are:

$$\begin{aligned}
P_1 &= \{\langle 0.75, 76\rangle\} \\
P_2 &= \{\langle 0.5, 74\rangle, \langle 1.5, 69\rangle\} \\
P_3 &= \{\langle 0, 72\rangle, \langle 0.5, 74\rangle, \langle 0.75, 76\rangle\}
\end{aligned}$$

Finally, we determine the set of vectors from set W (see Equation 5.5) which will translate each pattern P within dataset D. This completes the SIATEC algorithm, and for our example dataset, the solution is:

$$\begin{aligned}
TEC_1 = \langle &\{\langle 0.75, 76\rangle\}, \\
&\{\langle 0, 0\rangle, \langle 1, -5\rangle, \langle -0.75, -4\rangle, \langle 0.75, -7\rangle, \langle 0.25, -9\rangle, \langle -0.25, -2\rangle\}\rangle \\
TEC_2 = \langle &\{\langle 0.5, 74\rangle, \langle 1.5, 69\rangle\}, \\
&\{\langle 0, 0\rangle, \langle -0.5, -2\rangle, \langle 0.25, 2\rangle\}\rangle \\
TEC_3 = \langle &\{\langle 0, 72\rangle, \langle 0.5, 74\rangle, \langle 0.75, 76\rangle\} \\
&\{\langle 0, 0\rangle, \langle 1, -5\rangle\}\rangle
\end{aligned}$$

5.2 Requirements

In this section, we translate the mathematical definitions of the SIA and SIATEC algorithm functions into a list of requirements which must be fulfilled by our Semantic Web implementation. Both algorithms share many common steps, and consequently our Semantic Web implementation takes the form of one complete procedure which produces the required output from both algorithms.

It is clear from Section 5.1.2 that we need to be able to represent multidimensional datapoints and vectors as RDF triples. We also need the ability to determine equality between vectors (and also therefore datapoints, which can be considered as a subclass of vectors), as well as implement greater than and less than vector comparisons. Furthermore, we need methods for representing and comparing members of sets V and W (see Equations 5.4 and 5.5 respectively), alternatively referred to as Vector Table Elements (VTEs), and of representing MTPs and TECs as RDF triples, as well as determining equivalence between those entities. The challenge here is in finding general solutions which can be applied to vectors of any number of dimensions, as well as MTPs and TECs consisting of any number of datapoints (which will in turn have some unknown number of dimensions themselves). Thus, our complete requirements are:

1. Represent multidimensional datapoints and vectors as RDF triples.
2. Determine greater than and less than relationships between the RDF representations of any two k-dimensional datapoints d_1 and d_2.
3. Order, using the relations defined in 2, a set of n (RDF) datapoints.

4. Calculate the vector $v = d_2 - d_1$ where d_1 and d_2 are RDF representations of datapoints and v is an RDF representation of a vector.
5. Compute the vector tables V and W, and represent them using RDF.
6. Order the elements of the vector tables V and W.
7. Determine equality between the RDF representations of any two k-dimensional vectors v_1 and v_2.
8. Compute MTPs and represent them as RDF.
9. Compute TECs and represent them as RDF.

5.3 A Semantic Web Implementation of the SIA and SIATEC Algorithms

In this section we describe our solutions to the requirements listed in Section 5.2, cross-referencing with the code listed in Appendix B. The code is also available online[6].

5.3.1 Requirement 1

Represent Multidimensional Datapoints and Vectors as RDF Triples

In order to satisfy requirement 1, we begin by defining some OWL classes and properties in a namespace `sia`:

```
sia:Vector          rdf:type  owl:Class .
sia:dimVal          rdf:type  rdf:Property .
sia:DimensionValue  rdf:type  owl:Class .
sia:dimension       rdf:type  rdf:Property .
sia:value           rdf:type  rdf:Property .
```

With these simple entities we may associate one or more `sia:DimensionValue` nodes with a single `sia:Vector` node. Each `sia:DimensionValue` node must have exactly one `sia:dimension` property and one `sia:value` property. Figure 5.5 illustrates how we would express the two-dimensional vector $[0, 72]$ using this ontology (notice that we have two `sia:DimensionValue` nodes – one to denote that the value of dimension 1 is 0, and another one to denote that dimension

[6] `https://code.soundsoftware.ac.uk/projects/siasesame/repository/entry/queries/sample.sparql`

2 has a value of 72). The full SIA ontology, including relevant property range assertions, is:

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sia:     <http://example.org/sia#> .

sia:Dataset a owl:Class.

sia:DimensionValue a owl:Class;
  owl:intersectionOf (
    [ a owl:Restriction;
      owl:onProperty sia:dimension;
      owl:cardinality 1]
    [ a owl:Restriction;
      owl:onProperty sia:value;
      owl:cardinality 1] ) .

sia:VectorTableElement a owl:Class .
sia:Vector a owl:Class .
sia:Datapoint rdfs:subClassOf sia:Vector.
sia:SetW a owl:Class .
sia:SetV rdfs:subClassOf sia:SetW .
sia:OrderedSet rdfs:subClassOf sia:Dataset .

sia:dimVal a rdf:Property .
sia:dimVal rdfs:range sia:DimensionValue .
sia:dimension a rdf:Property .
sia:value a rdf:Property .
sia:fromDatapoint a rdf:Property .
sia:toDatapoint a rdf:Property .
sia:memberOfOrderedSet a rdf:Property .
sia:memberOfOrderedSet rdfs:range sia:OrderedSet .
sia:vector a rdf:Property .
sia:vector rdfs:range sia:Vector .
sia:canBeTranslatedBy a rdf:Property .
```

Note though that our initial dataset will consist of datapoints, not vectors. We consider a `sia:Datapoint` to be a sub-class of a `sia:Vector` (a datapoint is simply a vector whose starting point is the origin O).

If we label the datapoints from the example score shown in Figure 5.2 using the letters A to F, as shown in Figure 5.6, then datapoint A could be represented

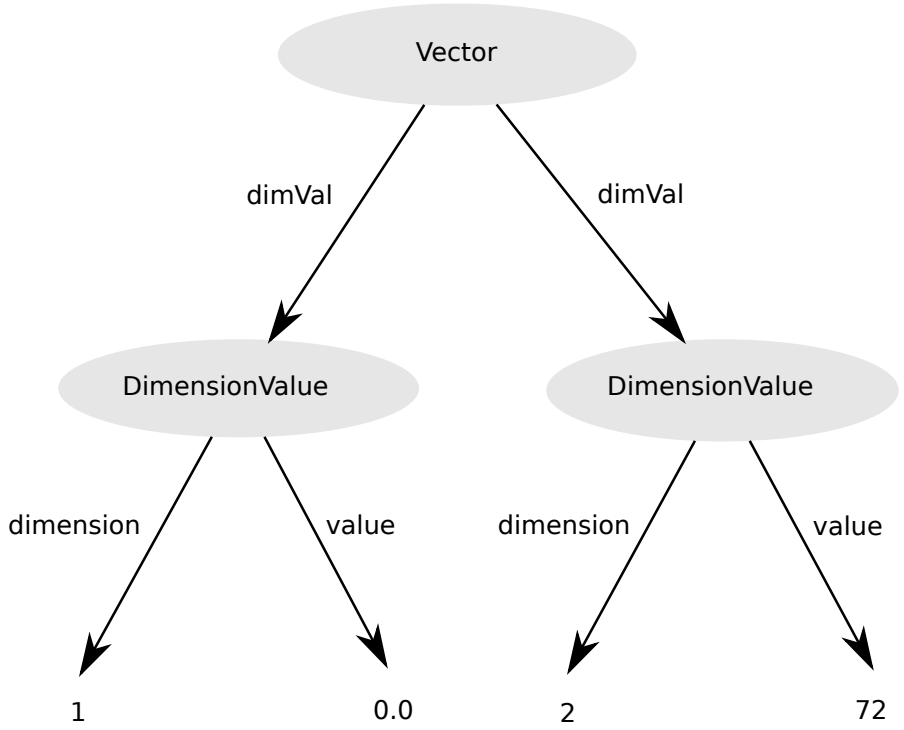

Figure 5.5: Conceptual representation of the vector $[0, 72]$, using *triples*

in RDF, using N3 notation,[7] as follows:

```
sia:A a sia:Datapoint;
      sia:vector _:vectorA;
      sia:memberOfDataset _:dataset1 .

_:vectorA sia:dimVal _:dva;
          sia:dimVal _:dvb .

_:dva sia:dimension "1"^^xsd:integer;
       sia:value "0.0"^^xsd:double .

_:dvb sia:dimension "2"^^xsd:integer;
        sia:value "72.0"^^xsd:double .
```

Everything preceded by `sia:` in this RDF data belongs to the namespace `sia`, whilst everything preceded by `_:` is a blank node. We have a node `sia:A`, which is a `sia:Datapoint`. Additionally, node `sia:A` has a `sia:vector` property, the object of which is the blank node `_:vectorA`, and a `sia:memberOfDataset`

[7] `http://www.w3.org/2000/10/swap/Primer.html`

property, the object of which is the blank node `_:dataset1`.

The blank node `_:vectorA` has two `sia:dimVal` properties – the object of one of them is the blank node `_:dva`, and the object of the other is the blank node `_:dvb`.

Blank node `_:dva` has a `sia:dimension` property, the object of which is the the integer value 1, and a `sia:value` property, the object of which is the double value 0.0. Blank node `_:dvb` has a `sia:dimension` property, the object of which is the the integer value 2, and a `sia:value` property, the object of which is the double value 72.0. Hence, datapoint A has the value 0 in dimension 1, and 72 in dimension 2.

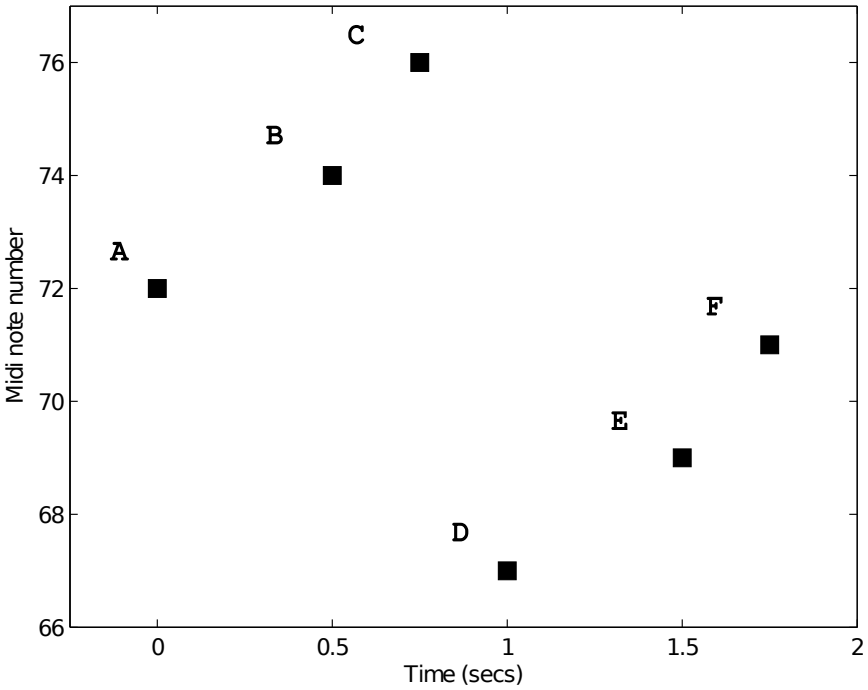

Figure 5.6: Datapoints labelled A to F

5.3.2 Requirement 2

Datapoint Relational Operation

We have already seen how, using Equation 5.3, to determine whether or not one datapoint is less than another. We now re-write Equation 5.3 in terms

of our SIA ontology. For a k-dimensional dataset, with datapoints d_1 and d_2 represented in RDF as

$\texttt{d}_1$ `rdf:type sia:Datapoint`
$\texttt{d}_2$ `rdf:type sia:Datapoint`

we say $d_1 < d_2$ iff all of the following RDF triples exist $\forall j \in \{1, ..., k\}$ and $\forall i \in \{1, 2\}$:

$\texttt{d}_1$ `sia:vector` $\texttt{v}_1$
$\texttt{d}_2$ `sia:vector` $\texttt{v}_2$
$\texttt{v}_1$ `sia:dimVal` $\texttt{dv}_{1,j}$
$\texttt{v}_2$ `sia:dimVal` $\texttt{dv}_{2,j}$
$\texttt{dv}_{i,j}$ `sia:dimension j`
$\texttt{dv}_{i,j}$ `sia:value` $\texttt{x}_{i,j}$

AND the following statement is true when $\exists j \in \{2, ..., k\}$ and $\forall m \in \{1, ..., j-1\}$:

$$(\texttt{x}_{1,1} < \texttt{x}_{2,1}) \vee ((\texttt{x}_{1,j} < \texttt{x}_{2,j}) \wedge (\texttt{x}_{1,m} = \texttt{x}_{2,m}))$$

5.3.3 Requirement 3

Order a set of n (RDF) datapoints

Beyond determining that $d_1 < d_2$, we also want to satisfy requirement 3, i.e. order a set of n datapoints. To implement such an ordering algorithm in a procedural programming language is trivial, but we're deliberately limiting ourselves here to the Semantic Web technologies; namely, RDF, OWL 2 and SPARQL 1.1. The key operations we need to perform are ordering (of dimensions) and relational comparison (between values). Furthermore, due to our objective of being able to handle datasets of any dimensionality k, the number of RDF triples across which these operations must be performed varies according to both n and k. RDF is merely a data format, and offers no mechanism for numerical comparison. Similarly, OWL, whilst offering various set (union, intersection etc.) and logic (owl:equivalentClass etc.) operations, has no numerical comparison

mechanism either.

SPARQL, on the other hand, does provide these operations, and consequently our Semantic Web technology implementation of the SIA and SIATEC algorithms takes the form of a collection of SPARQL queries, to be used in sequence, operating on an RDF dataset expressed in terms of the SIA ontology.

The two SPARQL queries we use to perform datapoint ordering are listed in Section B.1 of Appendix B. Query 1, labelled 'InsertDatapointOrderBarOne', is a nested query, two levels deep. The innermost query is duplicated here, with line numbers:

```
1   SELECT ?datapoint1 ?datapoint2
2          (MIN(?dimension) AS ?smallestDimensionMin)
3   WHERE
4   {
5     ?datapoint1 sia:vector ?vector1 .
6     ?vector1 a sia:Vector .
7     ?datapoint1 a sia:Datapoint .
8     ?datapoint1 sia:memberOfDataset ?dataset .
9     ?vector1 sia:dimVal ?dimVal1 .
10    ?dimVal1 sia:dimension ?dimension .
11    ?dimVal1 sia:value ?value1 .
12
13    ?datapoint2 sia:vector ?vector2 .
14    ?vector2 a sia:Vector .
15    ?datapoint2 a sia:Datapoint .
16    ?datapoint2 sia:memberOfDataset ?dataset .
17    ?vector2 sia:dimVal ?dimVal2 .
18    ?dimVal2 sia:dimension ?dimension .
19    ?dimVal2 sia:value ?value2 .
20
21    FILTER (?value1 != ?value2) .
22   }
23   GROUP BY ?datapoint1 ?datapoint2
```

This sub-query selects all possible values of the variable `?datapoint1` (line 1), such that there exists at least one RDF triple having subject `?datapoint1` and property `sia:vector` (line 5). The objects of any triples matching this condition can be anything – and whatever objects are found, are stored in the variable `?vector1`. Furthermore, there must exist at least one triple having subject `?vector1`, property `rdf:type` ('`a`' is a frequently used abbreviation of `rdf:type`), and object `sia:Vector` (line 6). Line 7 stipulates that any values of the variable `?datapoint1` must also have an `rdf:type` of `sia:Datapoint`. So far then, have have ensured that any values of the variable `?datapoint1` do

indeed represent datapoints, and in `?vector1`, we hopefully have a link to further triples which will provide us with the actual values of our multidimensional datapoint.

Line 9 adds the condition that for any values of the variable `?vector1`, there must exist at least one triple having property `sia:dimVal`. We would expect the number of triples matching this condition to be equal to the dimensionality of our dataset. Accordingly, all objects of any triples found to match this condition are stored in another variable, `?dimVal1`. For each specific value of `?dimVal1`, lines 10 and 11 pick out for us the dimension (variable `?dimension`) and the value (variable `?value1`).

Looking again at line 1, we see that it also selects all possible values of another variable, `?datapoint2`. The conditions associated with `?datapoint2` mirror those of `?datapoint1`, and are defined in lines 13 to 19. A key point to note though is that together, lines 10 and 18, by stipulating the same variable name `?dimension` as their objects, ensure that the values `?value1` and `?value2` we obtain from the two datapoints `?datapoint1` and `?datapoint2`, are *from the same dimension*. Similarly, lines 8 and 16 ensure that both datapoints belong to the same `?dataset`.

Line 21 adds a SPARQL FILTER to the set of conditions. This filter stipulates that for every set of results returned by the query, the values of the variables `?value1` and `?value2` must not be equal. The consequence of this is that our result set currently consists of all possible pairs of datapoints, and the dimensions in which each pair of datapoints have unequal values. Line 2 (which is a continuation of the SELECT part of this SPARQL query), is known as an *aggregate* function, and works in conjunction with the GROUP BY clause in line 23. For each distinct pair of datapoints `?datapoint1` and `?datapoint2` (as specified within the GROUP BY clause) in the resultset, it determines the MIN (minimum) value of all the values of `?dimension`, and stores this minimum value in a new variable `?smallestDimensionMin`. Consequently, this query returns a resultset containing three columns of variable values, `?datapoint1`, `?datapoint2`, and `?smallestDimensionMin`. Each row of the resultset therefore shows us the smallest dimension in which the two datapoints differ in value. The output of this query, acting upon our example dataset, is shown below (for brevity we display the variable values in abbreviated form, rather than full

URIs):

```
datapoint1 | datapoint2 | smallestDimensionMin                          |
-----------------------------------------------
sia#F      | sia#B      | 1                     |
sia#F      | sia#C      | 1                     |
sia#F      | sia#D      | 1                     |
sia#F      | sia#E      | 1                     |
sia#F      | sia#A      | 1                     |
sia#B      | sia#F      | 1                     |
sia#B      | sia#C      | 1                     |
sia#B      | sia#D      | 1                     |
sia#B      | sia#E      | 1                     |
sia#B      | sia#A      | 1                     |
sia#C      | sia#F      | 1                     |
sia#C      | sia#B      | 1                     |
sia#C      | sia#D      | 1                     |
sia#C      | sia#E      | 1                     |
sia#C      | sia#A      | 1                     |
sia#D      | sia#F      | 1                     |
sia#D      | sia#B      | 1                     |
sia#D      | sia#C      | 1                     |
sia#D      | sia#E      | 1                     |
sia#D      | sia#A      | 1                     |
sia#E      | sia#F      | 1                     |
sia#E      | sia#B      | 1                     |
sia#E      | sia#C      | 1                     |
sia#E      | sia#D      | 1                     |
sia#E      | sia#A      | 1                     |
sia#A      | sia#F      | 1                     |
sia#A      | sia#B      | 1                     |
sia#A      | sia#C      | 1                     |
sia#A      | sia#D      | 1                     |
sia#A      | sia#E      | 1                     |
```

In this instance, all unequal datapoints differ in dimension 1, but that wouldn't always necessarily be the case. These results are then used by an outer query, shown here (we abbreviate the body of the innermost sub-query from above with '...'):

```
1   SELECT ?datapoint1 (COUNT (?datapoint2) AS ?numSmallerDatapoints)
2          ?dataset
3   WHERE
4   {
5     ?datapoint1 sia:vector ?vector1 .
6     ?vector1 a sia:Vector .
7     ?vector1 sia:dimVal ?dimVal1x .
8     ?datapoint1 sia:memberOfDataset ?dataset .
9     ?dimVal1x sia:dimension ?smallestDimensionMin .
10    ?dimVal1x sia:value ?value1x .
11
12    ?datapoint2 sia:vector ?vector2 .
13    ?vector2 a sia:Vector .
14    ?vector2 sia:dimVal ?dimVal2x .
15    ?datapoint2 sia:memberOfDataset ?dataset .
16    ?dimVal2x sia:dimension ?smallestDimensionMin .
17    ?dimVal2x sia:value ?value2x .
18
19    FILTER (?value1x > ?value2x) .
20
21    {
22      SELECT ?datapoint1 ?datapoint2
23             (MIN(?dimension) AS ?smallestDimensionMin)
24      WHERE
25      {
26        ...
27      }
28      GROUP BY ?datapoint1 ?datapoint2
29    }
30  }
31  GROUP BY ?datapoint1 ?dataset
```

This query imposes much the same conditions upon `?datapoint1` and `?datapoint2` as before, with the exceptions that this time, we use the value of `?smallestDimensionMin` from the sub-query to specify in which dimension we want to query values `?value1x` and `?value2x` (lines 9, 10, 16 and 17), and we use a different FILTER condition – this time, we stipulate that the value of `?value1x` must be greater than that of `?value2x`. We also use the COUNT aggregate function, in conjunction with the GROUP BY clause of line 31, to count the number of `?datapoint2` datapoints for which `?value1x` (from `?datapoint1`) is greater than that of `?value2x` (from `?datapoint2`). We store this COUNT value in a new variable, `?numSmallerDatapoints`. The results now are as follows:

```
datapoint1 | numSmallerDatapoints | dataset |
-----------------------------------------------
sia#F      | 5                    | _:node1 |
sia#B      | 1                    | _:node1 |
sia#C      | 2                    | _:node1 |
sia#D      | 3                    | _:node1 |
sia#E      | 4                    | _:node1 |
```

The outermost part of the full query then uses a SPARQL BIND operation to increment the value of `?numSmallerDatapoints`. The result of this BIND operation is stored in a new variable `?orderedIndex`, and we are now able to use a SPARQL INSERT query to insert new triples into the triple store, indicating that each value of `?datapoint1` has a certain `?orderedIndex`, and is also a member of the ordered set `?dataset`. The outermost part of the query follows (again, we abbreviate the sub-queries listed above for brevity).

```
INSERT { ?datapoint1 sia:orderedIndex ?orderedIndex;
                     sia:memberOfOrderedSet ?dataset }

WHERE
{
  {
    SELECT ?datapoint1 (COUNT (?datapoint2) AS ?numSmallerDatapoints)
           ?dataset
    WHERE
    {
      ...
    }
    GROUP BY ?datapoint1 ?dataset
  }

  BIND (?numSmallerDatapoints + 1 AS ?orderedIndex)
}
```

Given n datapoints, this will assert the `sia:orderedIndex` of datapoints for indices 2 to n. Query 2 (Appendix B), 'InsertDatapointOrderLastOne', uses a FILTER NOT EXISTS condition to search for the sole remaining datapoint for which no `sia:orderedIndex` has been assigned, and assigns it a `sia:orderedIndex` of 1 accordingly. For our example dataset, the ascending order of the datapoints is A, B, C, D, E, F.

5.3.4 Requirements 4 and 5

Calculate the vector $v = d_2 - d_1$ where d_1 and d_2 are RDF representations of datapoints and v is an RDF representation of a vector

Compute the vector tables V and W, and represent them using RDF

The next step is to calculate the two vector tables, V and W. The elements of vector table W are all the vectors which connect every datapoint to every other datapoint in our dataset. We also retain a record of the pair of datapoints used to produce each vector – the 'from' datapoint and the 'to' datapoint. For our example datapoints A to F, vector table W is shown in Table 5.3. Vector table V is the subset of vector table W for which each 'from' datapoint is less than (in the sense defined in Section 5.3.2) the 'to' datapoint. For our example dataset, this gives us the vector table V shown in Table 5.2. Computing and asserting this vector table data in our RDF triple store is accomplished via five sequential SPARQL queries (see also Appendix B):

1. Query 3 'InsertSiatecVectorTableBnodes' – Creates skeleton entries for both vector tables (V and W), by selecting all possible pairs of datapoints `?datapoint1` and `?datapoint2` belonging to the same dataset, and creating one blank node of type `sia:VectorTableElement` for each pair. Asserts that this blank node has a `sia:fromDatapoint` which is `?datapoint1` and a `sia:toDatapoint` which is `?datapoint2`, and that it is associated with the same dataset as these datapoints. Note that we haven't yet determined the numerical values of the vectors themselves.

2. Query 4 'InsertSetVClassification' – Make use of the `sia:orderedIndex` property of the 'from' and 'to' datapoints associated with each `sia:VectorTableElement`, created in the previous step, in order to determine whether or not to assert that a particular `sia:VectorTableElement` has `rdf:type sia:SetV`.

3. Query 5 'InsertSetWClassification' – Assert that all `sia:VectorTableElement` subjects belonging to the same dataset have `rdf:type sia:SetW`.

4. Query 6 'InsertNewDimValsForVectorTable' – Determine which, if any, of the `sia:DimensionValue` nodes we will need to represent the numerical values in our vector tables, do not already exist in our triple store, and

create them. This simplifies future queries by avoiding duplicate RDF blank nodes which in fact represent the same thing.

5. Query 7 'InsertVectorTableDetails' – Now that all the necessary `sia:DimensionValue` nodes exist, make the correct associations between `sia:VectorTableElement` and `sia:DimensionValue` nodes.

5.3.5 Requirement 6

Order the Elements of the Vector Tables V and W

As with the datapoint ordering solution to Requirement 3, we employ two queries here – one (Query 8) which determines the `sia:orderedIndex` of `sia:VectorTableElement` nodes for indices 2 to n, and another (Query 9) for the remaining `sia:VectorTableElement` node. The larger part of the work is performed in Query 8, 'InsertVteOrderBarOne'. In order to determine whether one vector table element is greater than, equal to, or less than another, we first compare the two vectors (i.e. the vector values shown in Table 5.3) in the same way that we would compare two datapoints (see Section 5.3.2). In the cases where the two vectors are equal, we then perform the same relational operation on the 'from' datapoints of the two vector table elements.

Consequently, Query 8 is the UNION of two subqueries – one searches for all pairs of `sia:VectorTableElement` nodes, `?vte1` and `?vte2`, for which we are able to determine a greater than or less than relationship from the vector values alone, whilst the other searches for all pairs of `sia:VectorTableElement` nodes whose vector values are equal in all dimensions, and then compares the `sia:orderedIndex` value of the two corresponding 'from' datapoints. We now have a set of results showing us all the possible pairs of vector table elements where `?vte1` > `?vte2`. As with the datapoint ordering solution, we then determine the `sia:orderedIndex` of each `sia:VectorTableElement` by counting how many other `sia:VectorTableElement` nodes are 'less than' it.

5.3.6 Requirement 7

Determine equality between the RDF representations of any two k-dimensional vectors v_1 and v_2

Critical to later steps of the SIA and SIATEC algorithms is the ability to determine equality between vector table elements. In the context of vector table elements, by equality, we mean the same vector values, regardless of the 'from' and 'to' datapoints associated with the vector table element. So, for example, in our vector table V, shown in Table 5.2, the vector values of both the entries from A to D and B to E, are both (1, -5), hence we deem these vector table elements to be equivalent. As with several of the other requirements, the challenge is in finding a way to compare an unknown number of RDF object node values. Query 10 achieves this by selecting pairs of vector table elements having equal values in at least one dimension, counting the number of dimensions in which the vector table elements match, and filtering out any results for which this number is less than the dimensionality of the dataset. We then employ the OWL property `owl:equivalentClass` to assert that one vector table element `?vte1` is equivalent to another, `?vte2`. Because we are using an OWL-enabled reasoning engine, a consequence of this assertion is that triples of the form `?vte1 rdfs:subClassOf ?vte2` (as well as `?vte2 rdfs:subClassOf ?vte1`) will be added to our triple store, enabling us to easily locate equivalent vectors in future queries.

5.3.7 Requirement 8

Compute MTPs and represent them as RDF

The final step of the SIA algorithm (and also essential for the SIATEC algorithm) is to find all the MTPs for our dataset. We know from Section 5.1.2 that there is one MTP for each set of 'equivalent' vector table elements, and that the particular vector table element we want to select from each equivalent group is the 'smallest' one – i.e. the one having the smallest ordered index (calculated previously by Queries 8 and 9). Consequently, query 10 makes use of the `rdfs:subClassOf` relationship between equivalent vector table elements which we may now infer from the previous step, selecting only the vector table element `?vte` having the smallest `sia:orderedIndex` value from each equivalent

group, in order to assert that a new blank node `?mtp` has `rdf:type sia:Mtp` and `sia:vector ?vte`. We also select all the 'from' datapoints from the whole group of equivalent VTEs, and associate these with our new MTP – this is the group of datapoints which may be mapped onto another set of datapoints in our dataset via the vector values of the corresponding VTE. This completes the SIA algorithm.

5.3.8 Requirement 9

Compute TECs and represent them as RDF

Finally, completing the SIATEC algorithm, we find all the TECs for our dataset; that is, the unique set of datapoint patterns which are repeated at least once somewhere in our dataset, and the set of vectors which map each TEC to a new set of datapoints. We achieve this using two quite complex queries – Query 12 identifies the sets of MTPs which are in fact geometrically translated versions of each other, and selects from each group the one having the 'smallest' associated vector table element, which we now call a TEC. Query 13 then selects the set of vectors by which each TEC may be geometrically translated onto another MTP.

Both of these are non-trivial operations – Query 12 contains 8 levels of nested sub-queries. Beginning at the most deeply nested query and working outwards, we select the unique set of vectors from our vector table W. We then find all combinations of these vectors and all of our MTPs (also selecting all the datapoints belonging to each MTP), in order that we may test whether or not each MTP may be successfully mapped via every vector onto another MTP. We use a SPARQL BIND operation to project the value of one datapoint from each MTP, in one dimension, via the value of each vector, in the corresponding dimension, searching for matching datapoint values in other MTPs. Next, for each vector, MTP datapoint and projected MTP datapoint, we count the number of dimensions in which the two data points match, filtering out any cases in which the number of matching dimensions is not equal to the dimensionality of our dataset. From this set of data, we are able to count the number of datapoints that each MTP and each projected MTP have in common, before filtering out any results in which the number of matched datapoints is not equal to the total number of datapoints belonging to the MTP in question. The

results at this stage may still contain entries in which one vector was used to map between some pairs of datapoints, and a different one used for other datapoint pairs. Consequently we now count the number of vector table elements appearing per unique MTP pair, and filter out any pairs of MTPs for which this number is not one. We now have a list of distinct pairs of MTPs which map onto each other – this in turn enables us to group MTPs into equivalent sets. For each equivalent set, we select the MTP with the 'smallest' associated vector table element, and assert that this MTP also has `rdf:type sia:Tec`.

We have now added triples to our triple store which define our unique set of TECs, and it just remains to associate each TEC with the set of vectors from our vector table W which will translate the TEC onto other datapoints within our dataset. Starting again at the innermost sub-query and working outwards, Query 13 selects the unique set of vectors from vector table W, pairs each one with every TEC and concurrently counts the number of datapoints belonging to each TEC, performs the projection of the value of each dimension of each datapoint via each vector, counts the number of dimensions in which these projections successfully map onto other datapoints, counts the number of successfully projected datapoints and the number of datapoints belonging to the TEC, and filters out any results in which these numbers are not equal. We may then finally assert that a particular TEC can be translated by (using property `sia:canBeTranslatedBy`) certain VTEs. This completes the SIATEC algorithm.

5.3.9 Informative Queries

Queries 1 to 13, executed sequentially on a triple store containing datapoints described using the ontology shown in Section 5.3.1, carry out the necessary computations in order to find the MTPs and TECs which form the results of the SIA and SIATEC algorithms. Queries 14 to 18 may be used to extract these results – Query 14 shows us the full details of any MTPs found (the blank node representing the MTP, the ordered index of the vector by which this group of datapoints may be translated onto other datapoints, the values in each dimension of the vector, the values in each dimension of every datapoint belonging to this MTP, and the ordered index of each datapoint). Similarly, Query 15 shows us the details of any TECs found (the blank node representing

the TEC, the values in each dimension of the TEC's vector, and the values in each dimension of every datapoint belonging to this TEC). Queries 16 and 17 count the number of MTPs and TECs found, respectively, whilst Query 18 is a simple query, useful for debugging, which select all triples from the triple store. The full set of SPARQL queries are also available online[8], and we used the OWLIM RDF database management system[9] to run our queries.

5.3.10 MIDI to RDF

Before we are able to process a set of datapoints, we need some method of converting symbolic music data (in our case we use MIDI data) to the SIA RDF format described in Section 5.3.1. We use some custom written Java code, and classes from the javax.sound.midi package for extracting onset time, pitch and channel (three-dimensional) information from a midi file. In pseudo-code, the process is:

```
read midi file

for each midi track {

  get all midi events

  for each midi event {

    if the event is a note onset {

      get event time and pitch

      create an RDF blank node subject "datapointBnode" with
      property rdf:type and object sia:Datapoint

      find an existing, or create if one doesn't exist, RDF
      blank node "vectorBnode" which has three
      sia:DimensionValue objects corresponding to onset time
      (dimension 1), pitch (dimension 2) and channel number
      (dimension 3)

      Add the triple "datapointBnode" (subject) sia:vector
      (property) "vectorBnode" (object)
    }
  }
}
```

[8] https://code.soundsoftware.ac.uk/projects/siasesame/repository/entry/queries/sample.sparql

[9] http://www.ontotext.com/owlim

```
write the RDF dataset out to file
```

This process could easily be modified to incorporate information in other dimensions, e.g. offset time, channel loudness etc., or alternatively reduced down to two dimensions with just onset time and pitch (for example).

5.4 Performance Evaluation

There are two aspects to the evaluation of our SPARQL implementation of SIA and SIATEC – validation of the results, and execution time. In order to validate the results we compared our SPARQL results against both a Java implementation of SIA(TEC) written by the author of this thesis, and available from a Mercurial repository[10], and an implementation by Meredith, available from a subversion repository[11] using (a) the small set of two-dimensional data given in (Meredith et al., 2002), and (b) both two and three dimensional versions of a real midi file. In all cases the results matched.

Execution time of both our SPARQL implementation[12] and the purely Java version[10] was measured across a varying number of datapoints n, and for $k = 2$ and $k = 3$ dimensions. Both implementations were run on an Apple Mac Mini, with 2.3 GHz Intel Core i5 processor, 4GB 1333 MHz RAM, and default minimum and maximum heap size settings for the Java Virtual Machine. The results are shown with a log scale on the y-axis in Figures 5.7 ($k = 2$) and 5.8 ($k = 3$). We have a range of 1 to 150 datapoints for the Java implementation, but only 1 to 17 when $k = 2$ and 1 to 11 when $k = 3$ for the SPARQL implementation. Insufficient memory prevented us from obtaining results for higher numbers of datapoints for the SPARQL queries. Execution times for both implementations begin to exhibit polynomial and/or log type increases beyond certain datapoint number thresholds (approximately $n = 6$ for the SPARQL implementation and $n = 50$ for Java). We speculate that the slightly surprising shape of the curves below these thresholds are a consequence of the use of the 'Just-In-Time' compilation technique inherent in Java. Just-InTime compilation results in an initial

[10] `https://code.soundsoftware.ac.uk/hg/semantic-sia`

[11] `http://chromamorph.googlecode.com/svn/trunk`

[12] `https://code.soundsoftware.ac.uk/projects/siasesame/repository/entry/queries/sample.sparql`

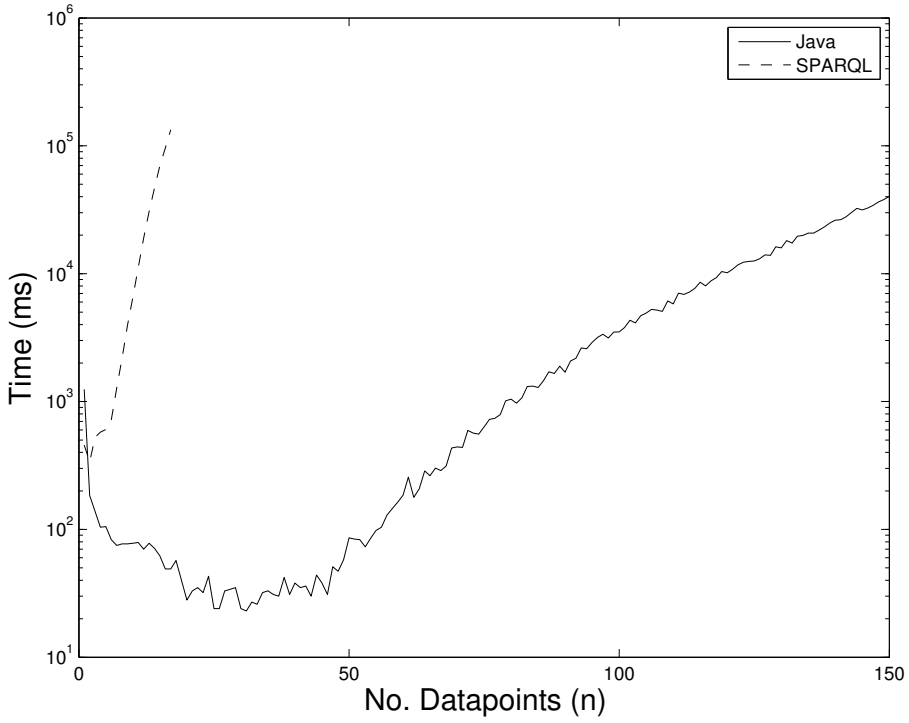

Figure 5.7: Execution time for both the Java and SPARQL implementations of SIATEC, for ($k = 2$) dimensions

'quick and dirty' compilation, which is fast to compile but not necessarily optimum in terms of execution time. The Java Virtual Machine then monitors the active threads at run time, and then if necessary, makes adjustments to the compiled code in order to optimise execution time. Our Java code involves a large number of calls to an object comparison method (for ordering datapoints and vectors), which may well be optimised by the Java Just-In-Time compiler as the number of datapoints increases.

Increasing the number of dimensions k from 2 to 3 has little effect on the execution time of the Java implementation, but is quite significant in the SPARQL implementation. The difference is shown more clearly in Figure 5.9. It's clear from Figures 5.7 and 5.8 that the execution time of the SPARQL version of SIATEC is significantly inferior to the pure Java version, reaching over 2 minutes for $n = 17$ datapoints and $k = 2$ dimensions, whereas the Java implementation takes just 40 seconds to process $n = 150$ two-dimensional datapoints.

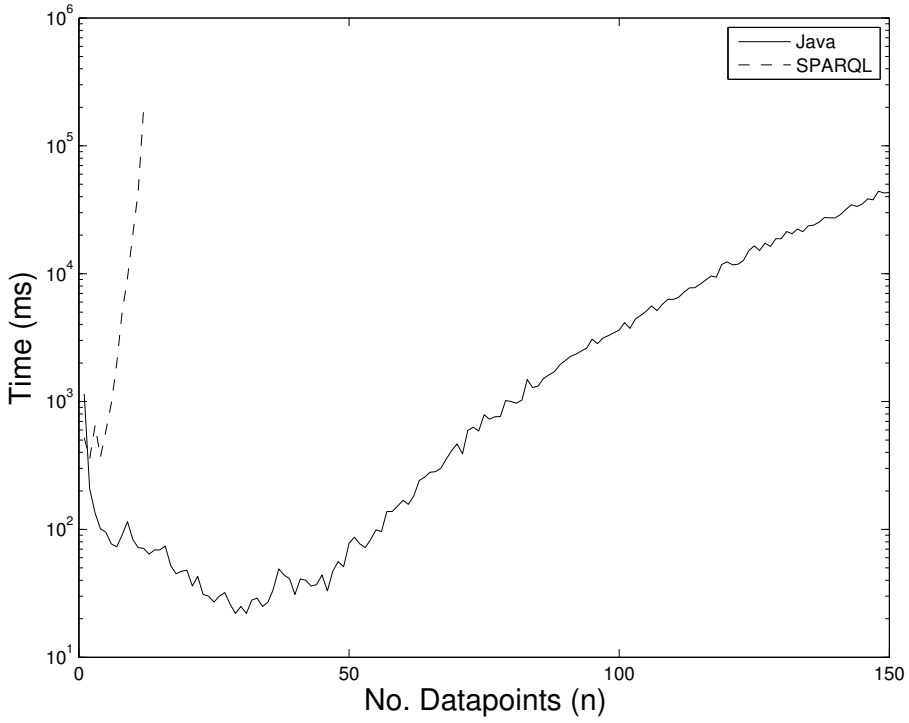

Figure 5.8: Execution time for both the Java and SPARQL implementations of SIATEC, for ($k = 3$) dimensions

Some heuristic curve-fitting reveals the execution time of our Java implementation to be approximately $\mathcal{O}(k^{0.3}n^6)$, compared to approximately $\mathcal{O}(k^2n^7)$ for our SPARQL implementation. Our SPARQL queries, then, introduce a significant performance cost, particularly with respect to the dimensionality k of the dataset, but also to the number of datapoints n.

The SPARQL implementation actually consists of thirteen separate queries executed in sequence. In Figure 5.10 we show a typical break-down (obtained for $n = 17$ datapoints and $k = 2$ dimensions) of the execution times of the individual queries as percentages of the full set of thirteen (for brevity we only show the slowest seven – the remaining queries have vanishingly small execution times compared to these seven). Already we observe that one particular query dominates the overall execution time; and it is instructive to delve a little deeper into the nature of each query in order to establish why some are faster than others.

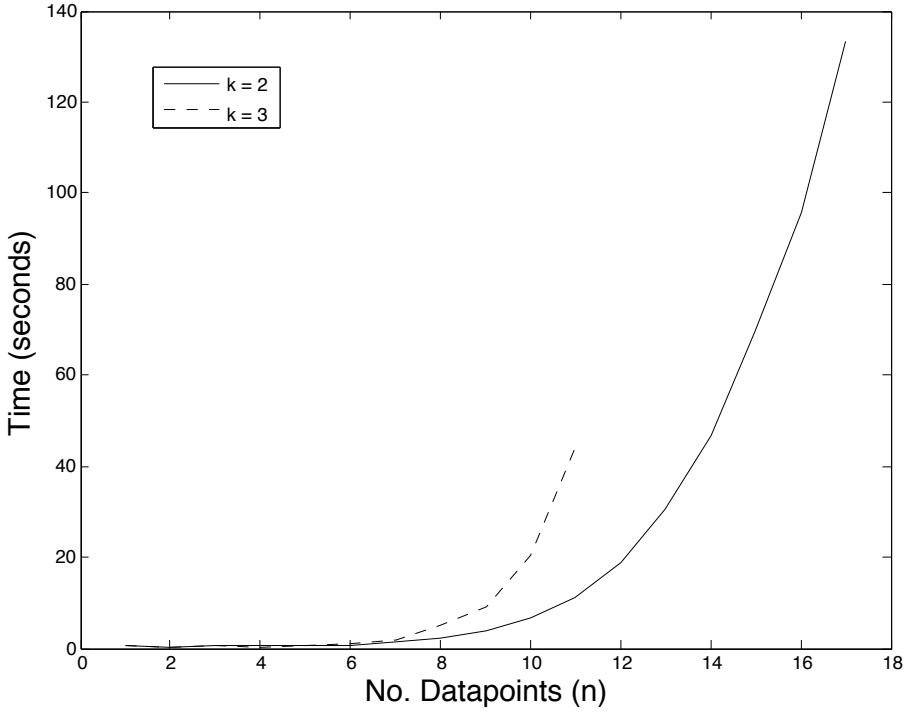

Figure 5.9: Execution time for the SPARQL implementation of SIATEC, for ($k = 2$ & $k = 3$) dimensions

SPARQL queries may involve various additional levels of complexity beyond the most simple cases. Our SIATEC queries, to differing degrees, utilise nested sub-queries, aggregate functions (e.g. find the minimum of a range of values), alternatives (the UNION SET operation), assignments (binding the result of an arithmetic operation to a variable), and restrictions (filtering out results according to certain conditions).

Table 5.5 makes explicit these additional levels of complexity for each query. The slowest query by far ('InsertDistinctTecs') also has the deepest level of nested sub-queries (eight), the highest number of aggregate functions (also eight), the joint-highest number of assignments (one), and the joint-second highest number of alternatives and restrictions (zero and three, respectively). Accordingly, the fastest of these seven queries ('InsertNewDimValsForVectorTable') has the joint lowest number in all categories except assignments.

The extensive use of nested sub-queries in combination with a high number

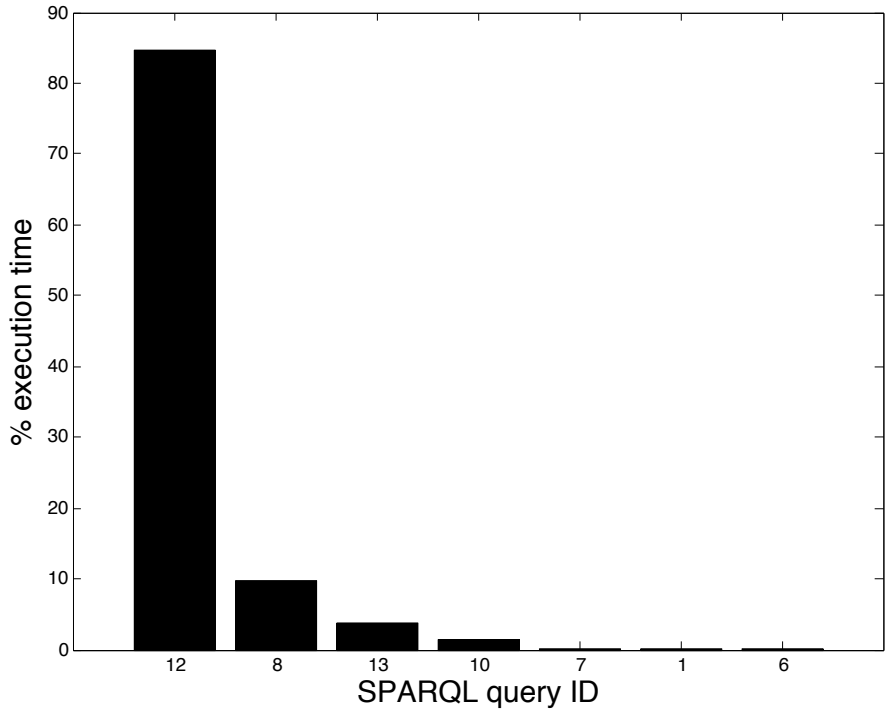

Figure 5.10: Relative execution times for SPARQL queries

Query ID	Query Name	% Execution Time	Deepest Nesting Level	Aggregate Functions	Alternatives	Assignments	Restrictions
12	InsertDistinctTecs	84.7	8	8	0	1	3
8	InsertVteOrderBarOne	9.8	3	4	1	1	5
13	InsertTecVectors	3.7	4	5	0	1	2
10	InsertVteEquivalence	1.5	0	2	0	0	3
7	InsertVectorTableDetails	0.2	0	0	0	1	3
1	InsertDatapointOrderBarOne	0.1	1	2	0	1	2
6	InsertNewDimValsForVectorTable	0.1	0	0	0	1	1

Table 5.5: SPARQL query execution times and complexity

of aggregate functions seems to present a clear performance cost, but is nonetheless difficult to avoid when attempting, as we are here, to perform comparisons between complex, multi-dimensional entities.

5.5 Evaluation with Respect to the New MIR Paradigm

In Chapter 3 we described our vision of a new MIR paradigm, based upon early, studio-based metadata capture, and exploitation of open, machine-readable Semantic Web data. We repeat here our list of fundamentally different aspects of the new paradigm, as well as the illustrative Semantic Audio Paradigm diagram (Figure 5.11) from Chapter 3. In the new paradigm, we:

- Exploit the processing power available to us in the recording studio
- Simplify the complexity and/or increase the accuracy of MIR algorithms by targeting source audio tracks rather than the full mix
- Are able to use the results of one MIR algorithm within the execution of another
- Produce a rich set of symbolic, or close to symbolic, metadata for a piece of recorded music
- Exploit the potential of the Semantic Web by publishing our metadata in a common, machine-readable, format
- Infer new musical information at a later date via less computationally expensive processing of our symbolic metadata (e.g. via the use of ontological inferencing or SPARQL queries)

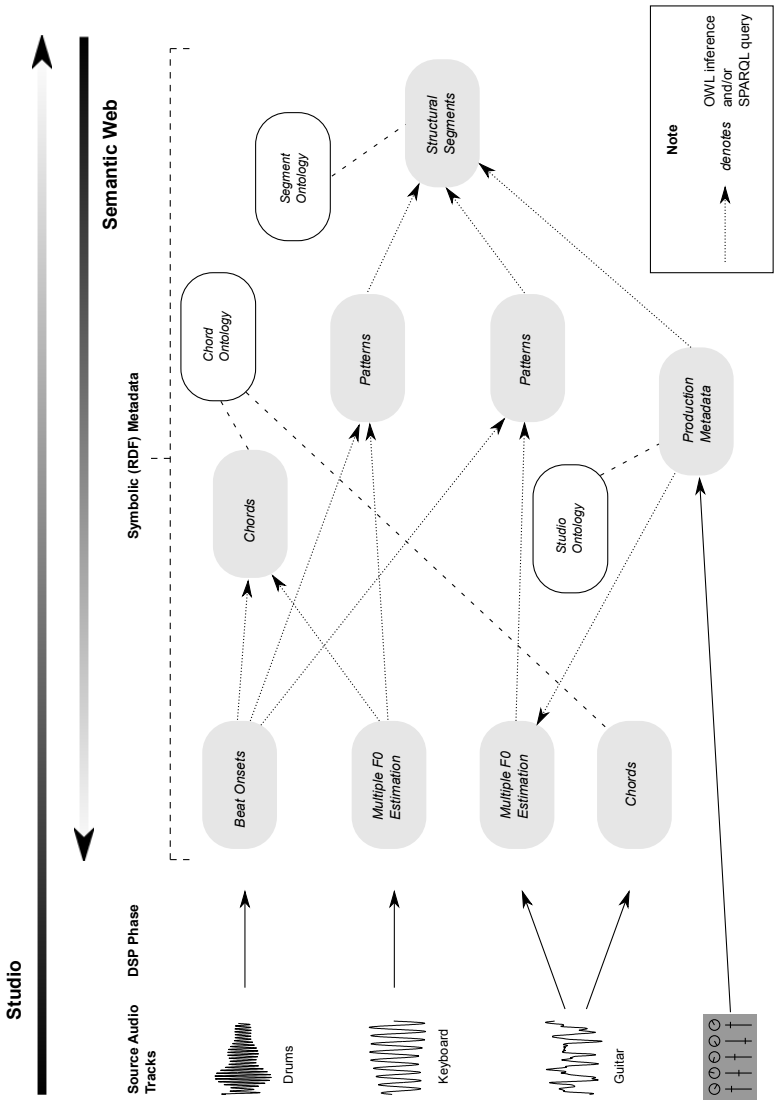

Figure 5.11: Semantic Audio Paradigm

Taking each item of this list in turn, we may evaluate our SPARQL version of SIA and SIATEC in the wider context of this new vision.

Exploit the processing power available to us in the recording studio

This is not directly applicable to our SPARQL SIA and SIATEC research, as the assumption here is that we have already performed some complex processing in the recording studio to produce the appropriate symbolic data which forms the input to our implementation (rather, this aspect is implicitly dealt with in Chapter 4).

Simplify the complexity and/or increase the accuracy of MIR algorithms by targeting source audio tracks rather than the full mix

Again, this point is more directly relevant to any earlier signal processing which may have occurred, such as that described in Chapter 4. Nevertheless it is important to acknowledge here that due to the limitations of current SPARQL implementations, we have introduced a significant performance degradation by using SPARQL rather than a more conventional, non-Semantic Web programming language. Additionally, the SPARQL queries used are themselves quite complex.

The ability to use the results of one MIR algorithm within the execution of another

The input (onset time and pitch) data for our implementation is assumed to have been generated by a signal processing-based algorithm earlier on in the chain. The output from our implementation, which is MTP and TEC data in RDF format, would certainly form valuable input data to other Semantic Web components of the overall picture, such as the inference of structural segments from perceptually significant pattern locations.

Production of a rich set of symbolic, or close to symbolic, metadata for a piece of recorded music

Together with the assumed existence of onset time and pitch symbolic data, our RDF MTPs and TECs form an additional and important part of the rich set

of symbolic data envisioned in Chapter 3, which could easily be augmented by RDF versions of chord, beat, segment and production metadata, for example.

Exploiting the potential of the Semantic Web by publishing our metadata in a common, machine-readable, format

We have successfully enabled the discovery of repeated patterns and their generation as RDF, fulfilling this aspect of the vision.

Infer new musical information at a later date via less computationally expensive processing of our symbolic metadata

We have partially fulfilled this requirement, in that we have successfully inferred the locations of repeated patterns from our input data via a set of SPARQL queries. However, the amount processing and memory required is, at this stage, too great. Nevertheless we assume that the performance of SPARQL engines as well as OWL reasoners will improve as Semantic Web technologies mature, and, combined with improved knowledge of the capabilities of the Semantic Web, we may move towards complete fulfilment of this requirement.

We also described some use-cases in Chapter 3, and whilst the aim of this present chapter was not to fulfil all of them, it is nevertheless instructive to assess what impact, if any, our SPARQL SIA and SIATEC implementation has in relation to those.

Semantic Navigation Around a Multitrack Audio Project

Repeated melodic patterns are precisely the kind of fundamental musical component we would like to visualise and navigate to within an audio project, and furthermore, they are often the mid-level sub-components of verse / chorus level structure. Consequently, our ability to both locate these patterns and also to label them as RDF data is a critical element in enabling improved semantic navigation around an audio project.

Custom End-User Audio Content

Our SPARQL implementation of SIA and SIATEC helps us to identify perceptually significant patterns, which moves us closer to fulfilment of this use-case.

Further audio processing and additional metadata would be required for a complete implementation.

Advanced Online Music Search

Again, the ability to discover repeat patterns provides us with a partial fulfilment of this use-case. We still require rhythm, chord, musician, instrumentation, production and musical idiom metadata. Nevertheless, semantic web ontologies do already exist for several of these types of metadata – e.g. chord[13], musician and instrumentation (Raimond et al., 2007), and production (Fazekas and Sandler, 2011; Fazekas, 2012).

Semantic Web Pattern Discovery

We have entirely satisfied this particular use-case, albeit with caveats about current execution time, as described in Section 5.4.

5.6 Discussion

We have successfully implemented a version of the SIA and SIATEC pattern discovery algorithms using purely Semantic Web technologies, opening up the possibility of performing many different MIR and musicological tasks in an internet-wide, machine architecture-independent manner. From a practical point of view however, we did find that the task was non-trivial compared to using an object-oriented programming language such as Java. Indeed, having completed this mostly SPARQL implementation of the algorithms, our impression is that we are very much pushing the boundaries of the type of computation which can be realistically achieved using SPARQL, OWL and RDF.

It is not possible to express the fact that a datapoint has value x in dimension i, y in dimension j, z in dimension k, and so on, up to some unknown number of dimensions, in a single triple. Instead, we resort to a hierarchy, making repeat use of blank nodes at each level of the hierarchy (as seen in Figure 5.5). However, the complexity of our data increases further – we also need to process VTEs, MTPs and TECs, which themselves 'contain' multiple datapoints and / or VTEs, and which we need to make relational comparisons between. One might

[13] `http://purl.org/ontology/chord/`

expect that OWL, dealing as it does with set membership and set operations, would be the ideal technology to utilise here – unfortunately though the types of data we need to express and compare are defined by their numerical values, and OWL does not possess the appropriate numerical operations. Whilst SPARQL does permit numerical operations, when attempting to query for this kind of complex, hierarchical structure, we frequently find that we can only query for one aspect of the structure at a time, and each of these aspects takes the form of a subquery to the next aspect for which we would like to query. So a typical query becomes a complex chain such as:

1. Select all vectors v_1
2. Select all MTPs mtp_1
3. Select all datapoints dp_1 belonging to all mtp_1
4. For each combination of vectors v_1, MTP mtp_1, datapoints dp_1, and dimension q, add the value of vector v_1 in dimension q to the value of datapoint dp_1 in the same dimension
5. For each combination of vectors v_1, MTP mtp_1 and datapoints dp_1, find, if one exists, the *DimensionValue* node belonging to a datapoint dp_2, itself belonging to an MTP mtp_2 which has the value $dp_1 + v_1$ in any one dimension
6. For each combination of vectors v_1, MTP mtp_1 and datapoints dp_1 and dp_2, count the number of dimensions $k_{projected}$ in which the projected datapoint dp_2 and the original datapoint dp_1 share the same value
7. Determine the dataset dimensionality k
8. Remove results for which $k_{projected} \neq k$
9. Determine whether each combination mtp_1 and mtp_2 share at least one datapoint
10. Count the number of datapoints s each combination mtp_1 and mtp_2 share
11. Count the number of datapoints n belonging to each mtp_1
12. Remove results for which $s \neq n$

13. Any remaining results satisfy our full search criteria

What would be a relatively simple task in (for example) an object-oriented programming language, quickly becomes considerably complex to express using Semantic Web technologies, and slow to execute. Furthermore, whilst we were able to find a suitable reasoning triple store in OWLIM, we found in general that deficiencies exist in certain other current alternatives.

Specifically, in the case of Pellet, queries which should have resulted in the insertion of one triple, the object of which was an xsd-typed literal, actually resulted in the insertion of two triples – both having the same subject and predicate, but one with an untyped literal and the other typed. This caused undesirable effects further down the query chain, and required the (theoretically unnecessary) use of SPARQL FILTERs to eradicate unwanted duplicate results. Another, perhaps more fundamental problem with Pellet, was that we found it would not allow RDF blank nodes as the subject of an owl:equivalentClass definition. OWL syntax does not prohibit this, and indeed, it is permitted by OWLIM. OpenLink Virtuoso was unable to process some of our more complex, deeply-nested SPARQL queries.

Despite these problems, we believe, given the significant amount of interest in the use of Semantic Web technologies within MIR (see Section 2.3.5), these are nevertheless important and useful results for other researchers working in the same field, providing valuable guidance regarding both the type of problems we might realistically expect to solve and the performance of taking such an approach.

Chapter 6

Conclusions and Further Work

6.1 Summary

In this thesis, partly in response to the evidence of a 'glass ceiling' for MIR algorithm accuracy, as well as the current level of interest surrounding the Semantic Web, we set out in Chapter 3 a vision for a new MIR paradigm. The vision is based around the principles of early, accurate signal processing-based methods of metadata generation in the recording studio, and the exploitation of open, machine-readable Semantic Web data as a means of sharing, querying, and making inferences from, lightweight symbolic music metadata. From this starting position, we then explored two important aspects of our new MIR paradigm – the potential to increase MIR algorithm accuracy via the use of multitrack audio (Chapter 4), and the implementation of an important class of pattern discovery algorithms using only Semantic Web technologies (Chapter 5).

When individual instrument recordings are mixed together, certain perceptually significant musical events which might have been prominent in their source tracks, occasionally become obscured by those occurring simultaneously in other tracks. Consequently, we hypothesised that certain MIR tasks could be made more accurate and/or simplified if we were to use multitrack audio as our input data, rather than the more typical case of fully mixed audio. The task of pitch detection for example, intuitively must surely become significantly more tractable when we are able to isolate one harmonic instrument at a time, and therefore also remove some potential sources of confusion. Separating instruments with different timbre, fundamental pitch and harmonics, as well as

percussive instruments with their inherently wide dynamic range and broad frequency spread, allows us to concentrate our focus on the specific characteristics of one particular source from an ensemble. Temporal trajectories of melody and harmony too, become easier to predict. We can of course make a similar case for beat-tracking concentrated solely on a percussion track, albeit there are cases in which some musical attribute may well be spread across multiple source tracks, and therefore only truly makes 'sense' when analysed as a whole.

One MIR task, that of structural segmentation, stood out as a particularly interesting case in this respect. A human listener can, and will, perceive the transition from one structural segment to another from a variety of cues (Bruderer, 2008), which may themselves originate from any subset of the entire musical ensemble. Moreover, if a particular cue is audibly prominent at regular intervals (a cymbal crash on beat one of every four bars, for example) but obscured by other instruments in certain passages, the listener may well still *infer* the same event, and perceive a structural boundary. If we happen to use an algorithm focusing on the detection of local, short time-span audio frame changes, then this event will be lost to some degree in the mixed audio recording, but still present in a multitrack version.

Consequently, we set out in Chapter 4 to test our hypothesis. We applied a typical segmentation technique, based upon audio feature extraction, self-distance matrices and homogeneity detection via novelty curves, to both mixed and multitrack versions of the same dataset, and compared the results. We also compared our results to those obtained using a state-of-the-art algorithm applied to mixed audio. In the process we created a publicly accessible, human-annotated, ground-truth structural segmentation dataset consisting of 104 multitrack rock and pop songs. Our results were very favourable, indicating a significant improvement in segmentation accuracy for multitrack audio when compared to mixed. Specifically, our method applied to multitrack data resulted in an F-value of 0.38, compared to 0.30 for mixed data, and 0.29 for the state-of-the-art algorithm applied to mixed data (at a 1s tolerance). We determined that results were optimal when we used equal weightings of RMS energy, chroma and MFCC features, and that the rhythmogram feature was of no benefit. Additionally, a comparison of two human segmentation annotations for the same song lent some weight to our prediction that there is an implicit limit to the level of confidence one may have in any human judgement

of structural segments.

It is undoubtedly the case that current MIR techniques, to varying degrees, still have room for improvement in terms of accuracy (see Section 2.1). Nevertheless, with the ability to target the single instrument recordings available in multitrack rather than mixed data, together with the ongoing development of MIR algorithms, it seems reasonable to assume that we are moving towards a situation in which we will be able to generate close to a full, accurate set of symbolic metadata (onsets, offsets, pitches, keys, chords, beats, tempos, and segments) from recorded multitrack audio. Crucially, by then publishing this metadata in a machine-readable, implementation-independent format, together with ontologies, we open up a vast amount of new possibilities in terms of further musicological analysis, as well as music data sharing and searching. The ability to go beyond simple linked-data and actually perform algorithmic processing of RDF data moves us closer to the prospect of not only representing, linking, and sharing music metadata, but also deriving new insights into musical content itself.

In Chapter 5, therefore, taking multichannel, symbolic (MIDI) data as our starting point, we investigated the viability of conducting further content analysis (in this case, pattern discovery) of existing RDF music metadata without having to leave the Semantic Web domain. We showed that it is technologically possible to fully implement a known pair of pattern discovery algorithms, SIA and SIATEC, using a combination of SPARQL queries, OWL, and RDF. However, the execution times of our Semantic Web implementation compared unfavourably with a more conventional (Java) programming language implementation. Execution time was found empirically to be (worst-case) $\mathcal{O}(k^2n)$ times worse than the Java version (where k is the dimensionality of the dataset and n is the number of datapoints), and required impracticably large amounts of computer memory.

We established that both expressing, and querying for, complex, compound data structures using Semantic Web technologies, whilst possible, is somewhat cumbersome and inefficient using currently available implementations of reasoning-enabled RDF data storage tools. We were able to produce a working implementation using OWLIM[1], but we also identified certain deficiencies in

[1] `http://www.ontotext.com/owlim`

other widely-used alternatives (e.g. Pellet and OpenLink Virtuoso) which were therefore unusable for our purposes.

6.2 Specific Contributions

- Empirical evidence that structural segmentation accuracy may be significantly improved by using multitrack rather than mixed audio recordings. This result was published in the IEEE Transactions on Audio, Speech and Language Processing journal (Hargreaves et al., 2012).
- A human-annotated structural segmentation ground-truth dataset of multitrack audio, containing 104 songs[2], publicly accessible for re-use by other researchers.
- Proof-of-concept evidence that a pattern discovery algorithm involving complex, compound data structures, can successfully be fully implemented using only Semantic Web technologies, together with performance evaluation metrics and full implementation details (Hargreaves et al., 2014, in print).

6.3 Future Work

In this thesis we have researched signal processing methods for locating segment boundaries in multitrack audio, and Semantic Web methods of pattern discovery in symbolic data. However we do not have to limit ourselves to these goals. To date, and to the authors' knowledge, all MIR research – be that structural segmentation, genre/artist/mood classification, music similarity measurement, onset/key detection, cover song identification, or chord/melody extraction, has been undertaken using either fully mixed music or single instrument recordings alone. The potential advantages offered by early capture of a more accurate and rich set of metadata from full multitrack sources in the studio are vast, and, because the metadata need not stay tightly bound to the commercial audio recording, are not limited to the improvement of studio editing tools. RDF data is already being used to enhance on-line artist information websites such as that

[2] `http://c4dm.eecs.qmul.ac.uk/rdr/handle/123456789/36`

provided by the BBC[3], and the publication of enhanced metadata alongside commercial audio recordings would only increase their versatility. Instead of concentrating on complex signal processing forms of 'reverse engineering' fully mixed audio, the MIR community may instead concentrate on exploiting an already present, easy to query and potentially vast amount of metadata via logical inferencing.

6.3.1 Instrument-Specific Audio Features for Structural Segmentation

Our first structural segmentation experiment, described in Section 4.4 of Chapter 4, used an algorithm based upon extraction of four audio features, self-distance matrix calculation, and homogeneity detection. It paid no attention to the particular type of instrument present in each source track. We then described a subsequent experiment in Section 4.5 in which we picked audio features according to the general class of musical instrument in each track; however, segmentation accuracy did not improve compared to the first experiment. This result is inconclusive – it may be the case that the particular instrument groupings and instrument group to audio feature mappings we used were simply not optimal. Current computing resources placed a limit on the number of instrument groupings and feature mappings we were able to test, but that does not preclude the possibility that given either a more sophisticated method of optimising the groupings and mappings, or simply increased brute-force compute power, we might achieve better results using such an approach.

6.3.2 Audio Feature Selection for Structural Segmentation According to Musical Function

Instead of select audio features according to musical instrument type, another possibility would be to select them according to what we might loosely call 'musical function'. It is of course true that audio tracks containing the same type of instrument can nevertheless vary widely in terms of timbre, dynamics, and pitch range. A guitar recording, for example, could in one instance be flamenco, incorporating high levels of percussive transients as the player taps

[3] `http://www.bbc.co.uk/music`

the body of the instrument, whilst in other instances be electric and overdriven, or acoustic chords, or classical and melodic. Treating these types of recordings similarly does not necessarily make sense – a more suitable approach might be to treat (for example) percussive, harmonic, rhythmic and ambient recordings as distinct groups.

6.3.3 Minimum Dataset Requirements

It has been implicitly assumed so far that at the point of analysis, all the source audio tracks required prior to producing the final mixdown are present. However, at intermediate stages of the recording process, only a subset of these tracks will exist. Another interesting direction for future work would therefore be to determine how accurately we are able to segment incomplete subsets of multitrack projects. Answering this question would also enable us to establish whether or not either a single or a minimal number of tracks of certain instrument combinations are sufficient for the derivation of an accurate segmentation, without needing to perform analysis of tracks which offer little or no new information. Given that DAW multitrack projects frequently contain around 8, 16 or 24 tracks, this would have the added benefit of greatly reducing computational complexity.

6.3.4 Lower-Level Segmentation

In addition to high-level verse/chorus type segmentations, we should also expect to be able to achieve lower (i.e. bar and beat) level segmentations. Possible ways to achieve this might be by analysis of the sub-structure of self-distance matrices (i.e. recursively analyse a self-distance matrix after first identifying high-level boundaries) or by using existing beat tracking or transcription algorithms, for example.

6.3.5 Additional Multitrack-Based MIR Experiments

We demonstrated improved structural segmentation accuracy in Chapter 4, but we could just as easily try to gain similar improvements for other MIR tasks. Onset, pitch, key, chord, beat, and tempo detection, as well as timbre analysis, instrument identification and automatic transcription are all excellent candidates for similar experiments using multitrack data.

6.3.6 Semantic Web Technique Optimisations

One of our goals when implementing the SIATEC algorithms was to retain the ability to deal with any dataset dimensionality. Scope does exist though to simplify these queries enormously if we decided instead upon a set number of dimensions at the outset, which, in practical terms, is not such an unreasonable compromise.

Additionally, the SPARQL implementation of SIATEC we have presented here is *one* Semantic Web implementation of SIATEC, but not necessarily the *optimal* version. Indeed, in an earlier implementation version we used the 'minus' operation in several of our SPARQL queries, and could only process $n = 8$ two-dimensional datapoints in 37 minutes. Performing a minus operation on two large datasets is computationally expensive; eradicating this operation improved performance significantly, and it is of course possible that further optimisations still exist.

Perhaps more importantly though, a fundamental difference between taking a SPARQL approach to algorithm implementation, and using a more conventional object-oriented programming language, is that whereas in an object-oriented language one may perform an operation such as "given two instances A and B of a certain class, compare their respective values of property C", in SPARQL we may only ask "given RDF nodes A and B, perform a search across the dataset for one or more RDF nodes for which the *set* of conditions C is true. This set of conditions can, and indeed is likely, to grow in complexity and involve multiple sub-queries as the complexity of the entities we are modelling increases. Nevertheless, this is no different to the kind of operation performed by logic and constraint programming languages such as Prolog, which benefit from highly mature and optimised search strategies.

Directions for future work include further optimisations of our SPARQL queries, testing alternative SPARQL engines, comparing execution time against a Prolog (or other logic programming language) implementation, or, more generally, identifying common bottlenecks in current SPARQL implementations which cause the kind of deeply-nested queries we use to perform so poorly. Although several authors have published research pertaining to SPARQL query optimisation (Schmidt et al., 2010) and the computational complexity of SPARQL (Pérez et al., 2009), these works do not relate to SPARQL version 1.1, features

of which we make extensive use of here (particularly aggregate functions, subqueries and projections – see Table 5.5).

A rule-based approach, such as the Semantic Web Rule Language[4] (SWRL), is also worthy of investigation, although many current SWRL libraries only provide partial implementations of the full specification – the 'list builtin', for example, which we would anticipate being a requirement, is a common omission. Anecdotal evidence suggests that current SWRL implementations are still some distance from constituting a practical solution; moreover, the SWRL is currently only a W3C proposal, rather than a final specification.

Of great value would be the ability to make compound objects (such as our vectors, MTPs and TECs) the subject of a SPARQL query, rather than having to build complex, deeply-nested queries. Having to do the latter effectively leaves the definition of the compound object to the query itself, rather than to the data model. Alternatively, the ability to express compound objects in OWL, and then define (and evaluate) operations such as equality and greater than / less than (perhaps by drilling down each entity's graph until we can establish that they both have the same structure and, where applicable, compare actual node values) would be a valuable and wide-reaching research area.

6.3.7 SIATEC as a Segmentation Method

We made extensive use of the SIATEC algorithm in Chapter 5 as a means of demonstrating the use of Semantic Web technologies in algorithm implementation. Regardless of implementation technology though, this algorithm, besides its intended use for pattern discovery, holds some promise as a method of structural segmentation. It is widely accepted that sequence repetition (see Section 2.2.5) is a strong indicator of high-level music structure. Consequently, we suggest there may be great value in using the SIATEC algorithm as a basis for determining the structural segments within symbolic music data.

6.3.8 A Musical Affect Ontology

Lerdahl and Jackendoff (1996) put forward the argument that music itself is a purely psychological phenomenon, and, building upon this assertion, Wiggins (2009) argues that any attempt to derive meaningful information from a

[4] `http://www.w3.org/Submission/SWRL/`

recording of a musical performance must begin with an attempt to understand the relationship between these representations of music and the actual musical effects they stimulate in the human brain.

In many fields of scientific or philosophical study, in order to develop our understanding of some particular domain, we begin with a study of the language we commonly use to express and describe as many aspects of that domain as possible. Whilst it is true that music itself is not a branch of linguistics (when listening to music, we don't consciously generate a stream of words to describe what we are experiencing), we can nevertheless, to a certain degree, retrospectively and introspectively describe a piece of music linguistically. We could use various adjectives (e.g. aggressive, placid, sparse, dense, spiritual) to describe certain passages or indeed the whole piece, state that we experience some kind of "lift" at certain points, identify themes and patterns, or ascribe similes (e.g. "sounds like thunder").

In a task such as structural segmentation, we commonly use words such as verse and chorus when referring to certain passages of music. We also make associations between words; for example we might say that a new segment follows a crescendo, or an orchestral swell, which itself sounds rich and warm. Equally though, we wouldn't strongly associate the term "resolved" with "meter". These are just a few examples though of a very rich set of terms commonly used not only to describe the surface level aspects of music, but also our understanding of it and the emotional effects it has upon us. If these concepts do indeed describe our psychological reaction to, understanding of, and relationship with, music, then the fundamental purpose of MIR must be the ability to automatically arrive at the same kinds of descriptions from a starting point of digital audio signals. To this end, we propose the development of a new ontology, one of "Musical Affect". The advantage of an ontology over a more straightforward classification system, such as a taxonomy, is that we may elucidate the complexities and subtleties of the domain in depth as well as in relation to other aspects of the world.

Semantic web ontologies, in fact, in the MIR world, are nothing new; music (Raimond et al., 2007), audio features[5], studio (Fazekas and Sandler, 2011),

[5] `http://motools.sourceforge.net/doc/audio_features.html`

segment (Fields et al., 2011) and chord[6] ontologies already exist, and integrate with other ontologies not specifically related to music (the timeline[7] and event[8] ontologies, for example). Whilst all of these certainly have, to varying degrees, some relevance to and overlap with musical affects, none of them were designed with the specific objective of encapsulating the kind of vocabulary we use in that domain. Rather, they focus variously on specific aspects of music such as recording and performance, signal processing results, music production in the recording studio, segmentation and the symbolic representation of chords.

We began in Chapter 5 to use Semantic Web technologies to identify some perceptually significant patterns in music. The creation of musical affect ontology, possibly in conjunction with a probabilistic reasoner (although to the author's knowledge, probabilistic reasoning within the Semantic Web is currently very much in its infancy) might allow us to go even further, possibly inferring, for example, musical semantics such as crescendo, 'triumphant return' (i.e. the return, at the conclusion of a piece of music, of a recurrent theme throughout the work), or fundamental motifs, from sequences and combinations of notes, chords, meter, and key.

Full development of such an ontology could entail a rigorous examination of the domain; some authors (Kolozali et al., 2011; Jordanous, 2010) working in related fields have employed automatic or semi-automatic methods of ontology generation. Jordanous (2010) for example performs a statistical comparison of the vocabulary used in a corpus of texts having a high degree of relevance to the domain (judged by number of citations, year of publication and author), with a corpus representing more general British word usage (the British National Corpus[9]) in order to discover terms with a high degree of relevance within the domain of interest.

Consequently we propose a similar analysis, building upon our own domain knowledge and intuitions by examining relevant texts from musical emotion, music perception and cognition, and musicology.

[6] http://motools.sourceforge.net/chord_draft_1/chord.html
[7] http://motools.sourceforge.net/timeline/timeline.html
[8] http://motools.sourceforge.net/event/event.html
[9] http://www.natcorp.ox.ac.uk/

6.4 Applications

Beyond the principal results and conclusions presented in this thesis, we describe in the following subsections some further potential applications of this research include.

6.4.1 Improved Navigation within Digital Audio Workstations for Recording Studio Engineers

Navigation within Digital Audio Workstations is generally a matter of scrolling backwards or forwards through audio files, with visualisations of the audio waveforms displayed on screen. The ability to manually add temporal labels is usually present, and any distinct sections (created, for example, using copy and paste type operations), can easily be jumped to. The addition of an automatic segmentation capability, either to subsets of audio tracks or the whole set, would undoubtedly enhance the usability of such software tools. The engineer would be able to jump instantly to the starts or ends of perceptually significant segments, from where he or she could audition or edit the desired passages of audio.

6.4.2 Guidance Regarding the Applicability of Semantic Web Technologies to Algorithmic MIR

We presented some evidence in Sections 2.3.4 and 2.3.5 of the growing trend towards the use of the Semantic Web, both generally and within the MIR community. Researchers new to the field may find an abundance of textbook tutorial material hailing the promise and capabilities of the Semantic Web, and in particular, the inferencing capabilities of OWL. Our Semantic Web pattern discovery research, described in Chapter 5, offers valuable insight not only into the technical effort and resources required for such an endeavour, but also its efficacy.

6.4.3 Automatic Transcription

Transcribing music, even manually, is a non-trivial task, requiring a high level of understanding and experience of music theory. No one uniquely accurate transcription exists for each musical work; rather, it is the job of the transcription

expert to sensibly group notes into chords, sequences into bars, and to apply appropriate time and key signatures, and ties between notes. One critical aspect of this process is forming an understanding of the high-level structure of a piece, and, as such, any accurate structural segmentation algorithm can provide an invaluable pre-processing step for automatic transcription systems. Mauch et al. (2009), for example, employ just such an approach in order to enhance the results of a chord transcription algorithm.

6.4.4 Audio Thumbnailing

Large digital music databases are now commonplace, particularly amongst commercial music retailers and streaming services. The ability to locate a short snapshot, which is in some sense maximally representative of a piece of music, is of great benefit when browsing such large collections (Bartsch and Wakefield, 2005; Burges et al., 2005; Chai and Vercoe, 2003; Levy et al., 2006). Segmentation is a valuable aid when attempting to find these snapshots, and as such, any segmentation algorithm offering increased levels of accuracy is highly applicable to audio thumbnailing.

6.5 Final Thoughts

Music Information Retrieval is now an extremely active and wide-reaching research area. The starting point is usually some representation of music – be that either an audio recording or symbolic data, before some form of analysis follows, leading to a certain type of music metadata. Often this takes place at the end of the music production chain, but as we have demonstrated here, earlier analysis can be beneficial.

This metadata is not always the end of the story though. As described in our wider vision for a new MIR paradigm in Chapter 3, the results of one audio-based MIR algorithm may then be combined with those of another, to be analysed further, perhaps this time in the symbolic domain. The ultimate end goal, we believe, should be the ability not only to derive accurate low-level metadata, but to be able to derive new insights from that low-level data – insights which will enable, for example, musicologists to better understand compositional techniques, or a music recommendation engine to determine similarity between songs.

Often, MIR algorithms are heavily signal processing-based and probabilistic, although certain rules may be implicit within them. On the other hand, some algorithms are much more explicitly grounded in rules-based logic, acting on symbolic data. Moreover, the increasing prominence of the Semantic Web encourages us to think about ways in which we might exploit logic-based knowledge representation. Neither approach, alone, is likely to yield an over-arching method of answering all of the many different types of question we might ask of an audio recording, or a musical score; music itself is not purely a logic or rules-based phenomena, any more than it is purely a manifestation of signal processing or probability. The combination of all of these strands though holds much promise with regard to the goal of computing and representing a rich semantics of music.

The MIR community has already achieved impressive advances in certain areas of this overall picture, and in this thesis we hope to have contributed in greater depth to the themes of early, low level metadata capture, and Semantic Web-based knowledge discovery and representation. There is still huge potential for further research in this direction though – our vision is not only to be able to accurately determine which notes occur where, or what the structural segments of a song are, but also to build our understanding of what music actually *is* into the very data itself – a form of *musical knowledge representation*, from which we may infer new knowledge from basic facts. Rather than us setting out to answer just one particular, narrowly-defined question, the metadata then, in a sense, takes on a life of its own, and 'speaks for itself' - for example revealing to us that a piece under consideration has a certain type of compositional form, or that it concludes with a rousing crescendo. We hope other MIR researchers will continue this philosophy by further exploiting multitrack audio data early in the production chain, as well as enhancing the level of sophistication and utility of both existing and new Semantic Web technologies.

Appendix A

Novelty Curve Peak Picking

An alternative method of picking peaks from a novelty score, proposed by Brennan (2010), consists of the following steps:

1. Calculate the standard novelty score using Equation 2.15, with a kernel size of 32.

2. Generate 6 more novelty scores with increasing degrees of smoothness, using zero-phase versions of the 6th order low-pass Butterworth design filters having normalised (with respect to half the sample rate) cutoff frequencies of 0.5, 0.25, 0.125, 0.1, 0.075 and 0.05. We will call the resulting novelty scores $n_1, n_2, ...n_7$, where n_1 is the original, unfiltered novelty score, n_2 has a normalised cutoff frequency of 0.5, and n_7 has a normalised cutoff frequency of 0.05

3. Despite using zero-phase filters, the low-pass filtered peaks might span several higher frequency peaks in the unfiltered novelty score. Compensate for this peak-smearing by comparing the peak locations of each novelty score with those of the novelty score two numbers lower (i.e. compare n_7 to n_5, n_6 to n_4, etc. In the case of n_2, compare to the original novelty score n_1). To make each pairwise comparison, we use the Dixon (2006) peak-picking method to locate the peaks of both novelty scores, then search six beats either side of each peak in the smoother of the two novelty scores (i.e. the one filtered with the lower cutoff frequency) for a matching peak in the other novelty score. This gives us seven alternative sets of peaks (including the peaks from the original unfiltered novelty score).

4. Further refine the peak locations by comparing each set with the peaks from the original novelty score (again, by searching six beats either side of each peak for a match in the original set).

5. For each peak in the original novelty score, count how many of the other six sets of peaks also contain the same peak.

6. Peaks appearing in six or more sets constitute the highest level set of segment boundary temporal locations, those appearing four times or more make up the mid-level segment boundaries, and peaks appearing in at least two of the sets go to make up the lowest level of segment boundaries.

Appendix B

SIATEC SPARQL Queries

B.1 Requirement 3 Queries

Executed sequentially in the order they appear here, the following two queries satisfy requirement 3 in Chapter 5, Section 5.2.

Query 1 - InsertDatapointOrderBarOne

```
^[InsertDatapointOrderBarOne]
PREFIX sia: <http://example.org/sia#>

INSERT { ?datapoint1 sia:orderedIndex ?orderedIndex;
           sia:memberOfOrderedSet ?dataset}

WHERE
{
  {
    # Select each datapoint, and count the number of
    # datapoints which are 'smaller' than it
    SELECT ?datapoint1 (COUNT (?datapoint2) AS ?numSmallerDatapoints)
           ?dataset
    WHERE
    {
      ?datapoint1 sia:vector ?vector1 .
      ?vector1 a sia:Vector .
      ?vector1 sia:dimVal ?dimVal1x .
      ?datapoint1 sia:memberOfDataset ?dataset .
      ?dimVal1x sia:dimension ?smallestDimensionMin .
      ?dimVal1x sia:value ?value1x .

      ?datapoint2 sia:vector ?vector2 .
      ?vector2 a sia:Vector .
      ?vector2 sia:dimVal ?dimVal2x .
      ?dimVal2x sia:dimension ?smallestDimensionMin .
      ?dimVal2x sia:value ?value2x .

      FILTER (?value1x > ?value2x) .
```

```
      {
        # Select all pairs of datapoints whose values differ
        # in at least one dimension, and find the smallest
        # of those dimensions in which the values differ
        SELECT ?datapoint1 ?datapoint2
               (MIN(?dimension) AS ?smallestDimensionMin)
        WHERE
        {
          ?datapoint1 sia:vector ?vector1 .
          ?vector1 a sia:Vector .
          ?datapoint1 a sia:Datapoint .
          ?datapoint1 sia:memberOfDataset ?dataset .
          ?vector1 sia:dimVal ?dimVal1 .
          ?dimVal1 sia:dimension ?dimension .
          ?dimVal1 sia:value ?value1 .

          ?datapoint2 sia:vector ?vector2 .
          ?vector2 a sia:Vector .
          ?datapoint2 a sia:Datapoint .
          ?datapoint2 sia:memberOfDataset ?dataset .
          ?vector2 sia:dimVal ?dimVal2 .
          ?dimVal2 sia:dimension ?dimension .
          ?dimVal2 sia:value ?value2 .

          FILTER (?value1 != ?value2) .
        }
        GROUP BY ?datapoint1 ?datapoint2
      }
    }
    GROUP BY ?datapoint1 ?dataset
  }

  # Use the number of datapoints which are smaller than
  # datapoint1 as the orderedIndex
  BIND (?numSmallerDatapoints + 1 AS ?orderedIndex)
}
```

Query 2 - InsertDatapointOrderLastOne

```
^[InsertDatapointOrderLastOne]
PREFIX sia: <http://example.org/sia#>
INSERT { ?datapoint sia:orderedIndex 1;
                    sia:memberOfOrderedSet ?dataset}

WHERE
{
  ?datapoint a sia:Datapoint .
  ?datapoint sia:memberOfDataset ?dataset .
  FILTER NOT EXISTS {?datapoint sia:memberOfOrderedSet ?orderedSet}
}
```

B.2 Requirements 4 and 5 Queries

Executed sequentially in the order they appear here, the following five queries (queries 3 to 7) satisfy requirements 4 and 5 in Chapter 5, Section 5.2.

Query 3 - InsertSiatecVectorTableBnodes

```
^[InsertSiatecVectorTableBnodes]
PREFIX sia: <http://example.org/sia#>
INSERT { _:vte rdf:type sia:VectorTableElement;
               rdfs:subClassOf _:vte;
               sia:fromDatapoint ?datapoint1;
               sia:toDatapoint ?datapoint2;
               sia:memberOfDataset ?dataset}
WHERE
{
  ?datapoint1 a sia:Datapoint .
  ?datapoint2 a sia:Datapoint .
  ?datapoint1 sia:memberOfOrderedSet ?orderedSet .
  ?datapoint2 sia:memberOfOrderedSet ?orderedSet .
  ?orderedSet a sia:OrderedSet

  {
    SELECT ?dataset ?orderedSet
    WHERE
    {
      ?orderedSet a sia:OrderedSet .
      BIND (bnode() AS ?dataset)
    }
  }
}
```

Query 4 - InsertSetVClassification

```
^[InsertSetVClassification]
PREFIX sia: <http://example.org/sia#>
INSERT { ?vte a sia:SetV }
WHERE
{
  ?vte a sia:VectorTableElement .
  ?vte sia:fromDatapoint ?datapoint1 .
  ?vte sia:toDatapoint ?datapoint2 .
  ?vte sia:memberOfDataset ?dataset .
  ?datapoint1 sia:orderedIndex ?i1 .
  ?datapoint2 sia:orderedIndex ?i2

  FILTER ((?datapoint1 != ?datapoint2) && (?i1 < ?i2))
}
```

Query 5 - InsertSetWClassification

```
^[InsertSetWClassification]
PREFIX sia: <http://example.org/sia#>
INSERT { ?vte a sia:SetW }
WHERE
{
  ?vte a sia:VectorTableElement .
  ?vte sia:memberOfDataset ?dataset .
}
```

Query 6 - InsertNewDimValsForVectorTable

```
^[InsertNewDimValsForVectorTable]
PREFIX sia: <http://example.org/sia#>
INSERT { _:dimVal a sia:DimensionValue;
                  sia:dimension ?dim;
                  sia:value ?val}
WHERE
{
  {
    SELECT DISTINCT ?dim ?val
    WHERE
    {
      ?vte rdf:type sia:VectorTableElement .
      ?vte sia:fromDatapoint ?datapoint1 .
      ?vte sia:toDatapoint ?datapoint2 .
      ?datapoint1 sia:vector ?vector1 .
      ?vector1 sia:dimVal ?dv1 .
      ?dv1 sia:dimension ?dim .
      ?dv1 sia:value ?val1 .
      ?datapoint2 sia:vector ?vector2 .
      ?vector2 sia:dimVal ?dv2 .
      ?dv2 sia:dimension ?dim .
      ?dv2 sia:value ?val2 .
      BIND (?val2 - ?val1 AS ?val)

      # Remove any pairs of dimensions and
      # values from our results for which
      # a sia:DimensionValue already exists
      FILTER NOT EXISTS
      {
        ?dimVal a sia:DimensionValue .
        ?dimVal sia:dimension ?dim .
        ?dimVal sia:value ?val
      }
    }
  }
}
```

Query 7 - InsertVectorTableDetails

```
^[InsertVectorTableDetails]
```

```
PREFIX sia: <http://example.org/sia#>
INSERT { ?vte sia:dimVal ?dimVal}
WHERE
{
  ?vte rdf:type sia:VectorTableElement .
  ?vte sia:fromDatapoint ?datapoint1 .
  ?vte sia:toDatapoint ?datapoint2 .
  ?datapoint1 sia:vector ?vector1 .
  ?vector1 sia:dimVal ?dv1 .
  ?dv1 sia:dimension ?dim .
  ?dv1 sia:value ?val1 .
  ?datapoint2 sia:vector ?vector2 .
  ?vector2 sia:dimVal ?dv2 .
  ?dv2 sia:dimension ?dim .
  ?dv2 sia:value ?val2 .
  BIND (?val2 - ?val1 AS ?val)
  ?dimVal a sia:DimensionValue .
  ?dimVal sia:dimension ?dim .
  ?dimVal sia:value ?val
}
```

B.3 Requirement 6 Queries

Executed sequentially in the order they appear here, the following two queries (queries 8 and 9) satisfy requirement 6 in Chapter 5, Section 5.2.

Query 8 - InsertVteOrderBarOne

```
^[InsertVteOrderBarOne]
PREFIX sia: <http://example.org/sia#>
INSERT { ?vte1 sia:orderedIndex ?orderedIndex;
         sia:memberOfOrderedSet ?dataset}

WHERE
{
  {
    # Select each VectorTableElement, and count the number of
    # VectorTableElements which are 'smaller' than it
    SELECT ?dataset ?vte1 (COUNT (DISTINCT(?vte2)) AS ?numSmallerVtes)
    WHERE
    {
      {
        SELECT ?vte1 ?vte2 ?dataset
        WHERE
        {
          ?vte1 sia:dimVal ?dimVal1x .
          ?dimVal1x sia:dimension ?smallestDimensionMin .
          ?dimVal1x sia:value ?value1x .

          ?vte2 sia:dimVal ?dimVal2x .
          ?dimVal2x sia:dimension ?smallestDimensionMin .
```

```
    ?dimVal2x sia:value ?value2x .

    FILTER (?value1x > ?value2x) .

    {
      {
        # Select all pairs of VectorTableElements
        # whose values differ in at least one dimension,
        # and find the smallest of those dimensions in
        # which the values differ
        SELECT ?vte1 ?vte2
               (MIN(?dimension) AS ?smallestDimensionMin)
               ?dataset
        WHERE
        {
          ?vte1 a sia:VectorTableElement .
          ?vte1 sia:dimVal ?dimVal1 .
          ?vte1 sia:memberOfDataset ?dataset .
          ?dimVal1 sia:dimension ?dimension .
          ?dimVal1 sia:value ?value1 .

          ?vte2 a sia:VectorTableElement .
          ?vte2 sia:dimVal ?dimVal2 .
          ?vte2 sia:memberOfDataset ?dataset .
          ?dimVal2 sia:dimension ?dimension .
          ?dimVal2 sia:value ?value2 .

          FILTER (?value1 != ?value2) .
        }
        GROUP BY ?dataset ?vte1 ?vte2
      }
    }
  }
}

UNION

{
  SELECT ?vte1 ?vte2 ?dataset
  WHERE
  {
    ?vte1 sia:fromDatapoint ?vte1FromDatapoint .
    ?vte1FromDatapoint sia:orderedIndex ?value1 .

    ?vte2 sia:fromDatapoint ?vte2FromDatapoint .
    ?vte2FromDatapoint sia:orderedIndex ?value2 .

    FILTER (?value1 > ?value2) .

    {
      # Select all pairs of VectorTableElements
      # whose vector values are equal in all dimensions
      SELECT ?dataset ?vte1 ?vte2
```

```
            WHERE
            {
              {
                SELECT ?dataset ?vte1 ?vte2
                       (COUNT (DISTINCT(?dimension)) AS ?numDims)
                WHERE
                {
                  ?vte1 a sia:VectorTableElement .
                  ?vte1 sia:memberOfDataset ?dataset .
                  ?vte1 sia:dimVal ?dimVal1 .
                  ?dimVal1 sia:dimension ?dimension .
                  ?dimVal1 sia:value ?value .

                  ?vte2 a sia:VectorTableElement .
                  ?vte2 sia:memberOfDataset ?dataset .
                  ?vte2 sia:dimVal ?dimVal2 .
                  ?dimVal2 sia:dimension ?dimension .
                  ?dimVal2 sia:value ?value .

                  FILTER (?vte1 != ?vte2) .
                }
                GROUP BY ?dataset ?vte1 ?vte2
              }

              {
                # Find the dimensionality of this dataset
                SELECT (COUNT (DISTINCT (?vectorDim))
                        AS ?datasetDimensionality)
                WHERE
                {
                  ?datapoint a sia:Datapoint;
                         sia:vector ?vector .

                  ?vector sia:dimVal ?vectorDimVal .

                  ?vectorDimVal sia:dimension ?vectorDim
                }
              }

              FILTER (?numDims = ?datasetDimensionality)
            }
            GROUP BY ?dataset ?vte1 ?vte2
          }
        }
      }
    }
    GROUP BY ?dataset ?vte1
  }
  BIND (?numSmallerVtes + 1 AS ?orderedIndex)
}
```

Query 9 - InsertVteOrderLastOne

```
^[InsertVteOrderLastOne]
PREFIX sia: <http://example.org/sia#>
INSERT { ?vte sia:orderedIndex 1;
              sia:memberOfOrderedSet ?dataset}
WHERE
{
  ?vte a sia:VectorTableElement .
  ?vte sia:memberOfDataset ?dataset .
  FILTER NOT EXISTS {?vte sia:memberOfOrderedSet ?orderedSet}
}
```

B.4 Requirement 7 Query

The following query (query 10) satisfies requirement 7 in Chapter 5, Section 5.2.

Query 10 - InsertVteEquivalence

```
^[InsertVteEquivalence]
PREFIX sia: <http://example.org/sia#>
INSERT {?vte1 owl:equivalentClass ?vte2}
WHERE
{
  {
    # Select all pairs of VectorTableElements whose
    # vector values are equal in all dimensions
    SELECT ?dataset ?vte1 ?vte2
           (COUNT (DISTINCT(?dimension)) AS ?numDims)
    WHERE
    {
      ?vte1 a sia:VectorTableElement .
      ?vte1 sia:memberOfDataset ?dataset .
      ?vte1 sia:dimVal ?dimVal1 .
      ?dimVal1 sia:dimension ?dimension .
      ?dimVal1 sia:value ?value .

      ?vte2 a sia:VectorTableElement .
      ?vte2 sia:memberOfDataset ?dataset .
      ?vte2 sia:dimVal ?dimVal2 .
      ?dimVal2 sia:dimension ?dimension .
      ?dimVal2 sia:value ?value .

      FILTER (?vte1 != ?vte2) .
    }
    GROUP BY ?dataset ?vte1 ?vte2
  }

  {
    # Find the dimensionality of this dataset
```

```
    SELECT (COUNT (DISTINCT (?vectorDim)) AS ?datasetDimensionality)
    WHERE
    {
      ?datapoint a sia:Datapoint;
                 sia:vector ?vector .

      ?vector sia:dimVal ?vectorDimVal .

      ?vectorDimVal sia:dimension ?vectorDim
    }
  }

  FILTER (?numDims = ?datasetDimensionality) .
  FILTER (?vte1 != ?vte2) .

}
```

B.5 Requirement 8 Query

The following query (query 11) satisfies requirement 8 in Chapter 5, Section 5.2.

Query 11 - InsertMtps

```
^[InsertMtps]
PREFIX sia: <http://example.org/sia#>
INSERT { ?mtp a sia:Mtp;
               sia:vector ?vte;
               sia:datapoint ?fromDatapoint }
WHERE
{
  {
    # Select every VectorTableElement ?vte, blank node
    # ?mtp, orderedIndex ?idx, and the minimum orderedIndex
    # ?minIdx of all VectorTableElements equivalent
    # to ?vte
    SELECT ?vte ?mtp ?idx (MIN (?allIdxs) AS ?minIdx)
    WHERE
    {
      {
        # Select every VectorTableElement of type
        # SetV, the vte ordered index, and a unique
        # blank node to be used as a maximally
        # translatable pattern (MTP)
        SELECT ?vte ?mtp ?idx
        WHERE
        {
          ?vte a sia:VectorTableElement;
               sia:orderedIndex ?idx;
               a sia:SetV .
          BIND (bnode() AS ?mtp) .
```

```
        }
      }

      # get the ordered indices of all
      # VectorTableElements which are
      # "equivalent" to ?vte (i.e. have
      # equal vector values in all dimensions)
      ?vteSub rdfs:subClassOf ?vte;
              a sia:VectorTableElement;
              sia:orderedIndex ?allIdxs;
              a sia:SetV .
    }
    GROUP BY ?vte ?mtp ?idx
  }

  # Restrict results to the VectorTableElement ?vte
  # having the smallest orderedIndex of the group
  # of VectorTableElements equivalent to ?vte
  FILTER (?idx = ?minIdx)
  ?vteSub rdfs:subClassOf ?vte .
  ?vteSub a sia:VectorTableElement .
  ?vteSub sia:fromDatapoint ?fromDatapoint .
}
```

B.6 Requirement 9 Queries

Executed sequentially in the order they appear here, the following two queries (queries 12 and 13) satisfy requirement 9 in Chapter 5, Section 5.2.

Query 12 - InsertDistinctTecs

```
^[InsertDistinctTecs]
PREFIX sia: <http://example.org/sia#>
INSERT {?tec a sia:Tec}
WHERE
{
  {
    # Select, from each group of Mtps which map onto other Mtps via
    # some VectorTableElement, the Mtp whose VectorTableElements has
    # the smallest orderedIndex, and call this Mtp ?tec (a
    # Translationally Equivalent Class)
    SELECT DISTINCT ?tec
    WHERE
    {
      {
        # Select the smallest orderedIndex of the VectorTableElements
        # of all the Mtps which Mtp ?mtp1 maps onto via some
        # VectorTableElement
        SELECT DISTINCT ?mtp1 (MIN (?mtp2Index) AS ?minMtpIndex)
        WHERE
        {
```

```
{
  # Select the number of VectorTableElements ?numVtes which
  # map ALL datapoints from Mtp ?mtp1 onto ALL datapoints
  # from Mtp ?mtp2 in ALL dimensions
  SELECT DISTINCT ?mtp1 ?mtp2
                  (COUNT (DISTINCT (?vte)) AS ?numVtes)
  {
    {
      # Find Mtps ?mtp2 for which we arrive at
      # ?numMatchedMtp1Datapoints datapoints from Mtp ?mtp1
      # via the vector values of VectorTableElement ?vte,
      # and count the number of datapoints ?numMtp1Datapoints
      # and ?numMtp2Datapoints belonging to ?mtp1 and ?mtp2
      SELECT DISTINCT ?vte ?mtp1
                      ?numMatchedMtp1Datapoints ?mtp2
                      (COUNT (DISTINCT (?mtp1Datapoint))
                        AS ?numMtp1Datapoints)
                      (COUNT (DISTINCT (?mtp2Datapoint))
                        AS ?numMtp2Datapoints)
      WHERE
      {
        {
          # Find Mtps ?mtp2 for which we arrive at at least
          # one datapoint from Mtp ?mtp1 via the vector
          # values of VectorTableElement ?vte, and count
          # the number of datapoints
          # ?numMatchedMtp1Datapoints and
          # ?numMatchedMtp2Datapoints for which the
          # vector mapping is valid
          SELECT DISTINCT ?vte
                          ?mtp1
                          (COUNT (DISTINCT (?mtp1Datapoint))
                            AS ?numMatchedMtp1Datapoints)
                          ?mtp2
                          (COUNT (DISTINCT (?mtp2Datapoint))
                            AS ?numMatchedMtp2Datapoints)
          WHERE
          {
            {
              # Find datapoints ?mtp2Datapoint belonging to
              # Mtp ?mtp2 which are arrived at from datapoint
              # ?mtp1Datapoint belonging to Mtp ?mtp1 via the
              # vector values of VectorTableElement ?vte, and
              # count the number of dimensions in which the
              # two datapoints match
              SELECT ?vte
                     ?mtp1
                     ?mtp1Datapoint
                     ?mtp2
                     ?mtp2Datapoint
                     (COUNT (DISTINCT (?dim)) AS ?numDims)
              WHERE
              {
```

```
{
  # Select all combinations of vtes (from the
  # previously selected set of "unique" vtes),
  # maximally translatable patterns ?mtp1,
  # and the datapoints belonging to ?mtp1
  SELECT ?vte
         ?mtp1
         ?mtp1Datapoint
  WHERE
  {
    {
      # Select the unique, in the sense
      # of having unique vector values,
      # set of VectorTableElements
      SELECT DISTINCT ?vte
      WHERE
      {
        {
          SELECT ?vte
                 (MIN (?vteSbIdx)
                  AS ?minVteIdx)
          WHERE
          {
            ?vte a sia:VectorTableElement;
                 rdfs:subClassOf ?vteSb .

            ?vteSb sia:orderedIndex ?vteSbIdx.
          }
          GROUP BY ?vte
        }

        ?vte sia:orderedIndex ?minVteIdx
      }
    }

    ?mtp1 a sia:Mtp;
          sia:datapoint ?mtp1Datapoint .
  }
}

?mtp1Datapoint sia:vector ?mtp1Vector .

?mtp1Vector sia:dimVal ?mtp1DimVal .
?mtp1DimVal sia:dimension ?dim .
?mtp1DimVal sia:value ?mtp1Val .

?vte sia:dimVal ?vteDimVal .

?vteDimVal sia:dimension ?dim .
?vteDimVal sia:value ?vteVal .

BIND (?mtp1Val + ?vteVal AS ?mtp2Val) .
```

```
                    ?mtp2 a sia:Mtp;
                          sia:datapoint ?mtp2Datapoint .

                    ?mtp2Datapoint sia:vector ?mtp2Vector .

                    ?mtp2Vector sia:dimVal ?mtp2DimVal .
                    ?mtp2DimVal sia:dimension ?dim .
                    ?mtp2DimVal sia:value ?mtp2Val .
                  }
                  GROUP BY ?vte ?mtp1 ?mtp1Datapoint
                           ?mtp2 ?mtp2Datapoint
                }

                {
                  # Find the dimensionality of this dataset
                  SELECT (COUNT (DISTINCT (?vectorDim))
                           AS ?datasetDimensionality)
                  WHERE
                  {
                    ?datapoint a sia:Datapoint;
                               sia:vector ?vector .

                    ?vector sia:dimVal ?vectorDimVal .

                    ?vectorDimVal sia:dimension ?vectorDim
                  }
                }

                FILTER (?numDims = ?datasetDimensionality)
              }
              GROUP BY ?vte ?mtp1 ?mtp2
            }

            ?mtp1 sia:datapoint ?mtp1Datapoint .
            ?mtp2 sia:datapoint ?mtp2Datapoint .
          }
          GROUP BY ?vte ?mtp1 ?numMatchedMtp1Datapoints ?mtp2
        }
        FILTER (?numMtp1Datapoints = ?numMtp2Datapoints
                && ?numMatchedMtp1Datapoints = ?numMtp1Datapoints)
      }
      GROUP BY ?mtp1 ?mtp2
    }

    FILTER (?numVtes = 1)

    ?mtp1 sia:vector ?vte1 .
    ?mtp2 sia:vector ?vte2 .

    ?vte1 sia:orderedIndex ?mtp1Index .
    ?vte2 sia:orderedIndex ?mtp2Index .
  }
  GROUP BY ?mtp1
```

```
      }

      ?tec a sia:Mtp;
           sia:vector ?mtpVector .
      ?mtpVector sia:orderedIndex ?minMtpIndex .
    }
  }
}
```

Query 13 - InsertTecVectors

```
^[InsertTecVectors]
PREFIX sia: <http://example.org/sia#>
INSERT {?tec sia:canBeTranslatedBy ?vte}
WHERE
{
  # Select every combination of TEC and VectorTableElement
  # where the TEC has a one-to-one mapping onto another MPT
  # via the VectorTableElement
  {
    SELECT ?vte ?tec ?numTecDatapoints
           (COUNT (DISTINCT (?projectedDatapoint))
            AS ?numProjectedDatapoints)
    WHERE
    {
      {
        # Select every combination of TEC and VectorTableElement,
        # along with the number of datapoints belonging to each TEC,
        # the actual datapoints belonging to each TEC, the
        # projectedDatapoint each TEC maps to via the
        # VectorTableElement in at least one dimension, and the number
        # of dimensions in which the TEC datapoint and the
        # projectedDatapoint match
        SELECT ?vte ?tec ?numTecDatapoints ?tecDatapoint
               ?projectedDatapoint
               (COUNT (DISTINCT (?dim)) AS ?numMatchedDims)
        WHERE
        {
          {
            # Select every combination of TEC and VectorTableElement,
            # along with the number of datapoints belonging to each TEC
            SELECT ?tec ?vte
                   (COUNT (DISTINCT (?tecDatapoint))
                    AS ?numTecDatapoints)
            WHERE
            {
              {
                # Select the unique, in the sense
                # of having unique vector values,
                # set of VectorTableElements
                SELECT DISTINCT ?vte
                WHERE
                {
```

```
            {
              SELECT ?vte (MIN (?vteSubIdx) AS ?minVteIdx)
              WHERE
              {
                ?vte a sia:VectorTableElement;
                     rdfs:subClassOf ?vteSub .

                ?vteSub sia:orderedIndex ?vteSubIdx.
              }
              GROUP BY ?vte
            }

            ?vte sia:orderedIndex ?minVteIdx
          }
        }

        ?tec a sia:Tec;
             sia:datapoint ?tecDatapoint .
      }
      GROUP BY ?tec ?vte
    }

    ?tec sia:datapoint ?tecDatapoint .

    ?tecDatapoint sia:vector ?tecVector .

    ?tecVector sia:dimVal ?tecDimVal .
    ?tecDimVal sia:dimension ?dim .
    ?tecDimVal sia:value ?tecVal .

    ?vte sia:dimVal ?vteDimVal .

    ?vteDimVal sia:dimension ?dim .
    ?vteDimVal sia:value ?vteVal .

    BIND (?tecVal + ?vteVal AS ?projectedVal) .

    ?projectedDatapoint a sia:Datapoint;
                        sia:vector ?projectedVector .

    ?projectedVector sia:dimVal ?projectedDimVal .
    ?projectedDimVal sia:dimension ?dim .
    ?projectedDimVal sia:value ?projectedVal .
  }
  GROUP BY ?vte ?tec ?numTecDatapoints
           ?tecDatapoint ?projectedDatapoint
}

# Find the dimensionality of this dataset
{
  SELECT (COUNT (DISTINCT (?vectorDim))
          AS ?datasetDimensionality)
  WHERE
```

```
        {
          ?datapoint a sia:Datapoint;
                     sia:vector ?vector .

          ?vector sia:dimVal ?vectorDimVal .

          ?vectorDimVal sia:dimension ?vectorDim
        }
      }

      FILTER (?numMatchedDims = ?datasetDimensionality)
    }
    GROUP BY ?vte ?tec ?numTecDatapoints
  }

  FILTER (?numTecDatapoints = ?numProjectedDatapoints)
}
```

B.7 Informative Queries

The queries below are do not form part of the Semantic Web implementation of the SIA and SIATEC algorithms, but are useful for extracting results.

Query 14 - SelectMtps

```
^[SelectMtps]
PREFIX sia: <http://example.org/sia#>
SELECT ?mtp ?idx ?dim ?val ?mtpElementIdx ?mtpDim ?mtpVal
WHERE
{
  ?mtp a sia:Mtp .
  ?mtp sia:vector ?mtpVector .
  ?mtpVector sia:orderedIndex ?idx .
  ?mtpVector sia:dimVal ?mtpDimVal .
  ?mtpDimVal sia:dimension ?dim .
  ?mtpDimVal sia:value ?val .
  ?mtp sia:datapoint ?datapoint .
  ?datapoint sia:vector ?vector .
  ?datapoint sia:orderedIndex ?mtpElementIdx .
  ?vector sia:dimVal ?vectorDimVal .
  ?vectorDimVal sia:dimension ?mtpDim .
  ?vectorDimVal sia:value ?mtpVal .
}
ORDER BY ?idx ?dim ?mtpElementIdx ?mtpDim
```

Query 15 - SelectTecs

```
^[SelectTecs]
PREFIX sia: <http://example.org/sia#>
SELECT ?tec ?tecDatapoint ?tecDatapointVectorDim
```

```
       ?tecDatapointVectorVal ?vector ?vectorDim ?vectorVal
WHERE
{
  ?tec a sia:Tec;
     sia:datapoint ?tecDatapoint;
     sia:canBeTranslatedBy ?vector.

  ?tecDatapoint sia:vector ?tecDatapointVector .

  ?tecDatapointVector sia:dimVal ?tecDatapointVectorDimVal .
  ?tecDatapointVectorDimVal sia:dimension ?tecDatapointVectorDim;
      sia:value ?tecDatapointVectorVal .

  ?vector sia:dimVal ?vectorDimVal .
  ?vectorDimVal sia:dimension ?vectorDim;
      sia:value ?vectorVal .
}
ORDER BY ?tec ?tecDatapoint ?tecDatapointVectorDim ?vector ?vectorDim
```

Query 16 - SelectCountMtps

```
^[SelectCountMtps]
PREFIX sia: <http://example.org/sia#>
SELECT (COUNT (DISTINCT (?mtp)) AS ?numMtps)
WHERE
{
  ?mtp a sia:Mtp
}
```

Query 17 - SelectCountTecs

```
^[SelectCountTecs]
PREFIX sia: <http://example.org/sia#>
SELECT (COUNT (DISTINCT (?tec)) AS ?numTecs)
WHERE
{
  ?tec a sia:Tec
}
```

Query 18 - SelectAll

```
^[SelectAll]
PREFIX sia: <http://example.org/sia#>
SELECT ?s ?p ?o
WHERE
{
  ?s ?p ?o
}
```

Bibliography

S. Abdallah, K. Noland, M. Sandler, M. Casey, and C. Rhodes. Theory and evaluation of a Bayesian music structure extractor. In *Proc. International Conference on Music Information Retrieval (ISMIR)*. Citeseer, 2005.

Dean Allemang and James Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Access Online via Elsevier, 2011.

ANSI. *USA standard acoustical terminology, Tech. Rep. S1.1-1960*. American National Standards Institute, 1960.

Grigoris Antoniou. *A semantic web primer*. the MIT Press, 2004.

J. Aucouturier, F. Pachet, and M. Sandler. "the way it sounds": timbre models for analysis and retrieval of music signals. *IEEE Transactions on Multimedia*, 7(6):1028–1035, 2005.

J.J. Aucouturier and M. Sandler. Segmentation of musical signals using hidden markov models. *Preprints-Audio Engineering Society*, 2001.

L. Barrington, A.B. Chan, and G. Lanckriet. Modeling music as a dynamic texture. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18 (3):602–612, 2010.

Mathieu Barthet, Steven Hargreaves, and Mark Sandler. Speech/music discrimination in audio podcast using structural segmentation and timbre recognition. In Sølvi Ystad, Mitsuko Aramaki, Richard Kronland-Martinet, and Kristoffer Jensen, editors, *Exploring Music Contents*, volume 6684 of *Lecture Notes in Computer Science*, pages 138–162. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23125-4.

M.A. Bartsch and G.H. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, 7(1):96–104, 2005.

Emmanouil Benetos and Simon Dixon. A temporally-constrained convolutive probabilistic model for pitch detection. In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2011 IEEE Workshop on*, pages 133–136. IEEE, 2011.

Emmanouil Benetos and Simon Dixon. Temporally-constrained convolutive probabilistic latent component analysis for multi-pitch detection. In *Latent Variable Analysis and Signal Separation*, pages 364–371. Springer, 2012.

Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: Breaking the glass ceiling. In *ISMIR*, pages 379–384, 2012.

A.S. Bregman. *Auditory scene analysis: The perceptual organization of sound*. The MIT Press, 1994.

T Brennan. Music structure analysis. Master's thesis, Queen Mary University of London, 2010.

Bertrand H Bronson. Mechanical help in the study of folk song. *The Journal of American Folklore*, 62(244):81–86, 1949.

M.J. Bruderer. *Perception and Modeling of Segment Boundaries in Popular Music*. PhD thesis, 2008.

M.J. Bruderer, M. McKinney, and A. Kohlrausch. Structural boundary perception in popular music. In *Proc. 7th Int. Conf. Music Inf. Retrieval*, pages 198–201. Citeseer, 2006.

C.J.C. Burges, D. Plastina, J.C. Platt, E. Renshaw, and H.S. Malvar. Using audio fingerprinting for duplicate detection and thumbnail generation. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 3. IEEE, 2005.

John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An expert ground truth set for audio chord recognition and music analysis. In *ISMIR*, pages 633–638, 2011.

Emilios Cambouropoulos. *Towards a General Computational Theory of Musical Structure*. PhD thesis, University of Edinburgh, UK, May 1998.

Chris Cannam, Mark Sandler, Michael O. Jewell, Christophe Rhodes, and Mark d'Inverno. Linked data and you: Bringing music research software into the semantic web. *Journal of New Music Research*, 39(4):313–325, 2010. doi: 10.1080/09298215.2010.522715.

W. Chai and B. Vercoe. Music thumbnailing via structural analysis. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 223–226. ACM New York, NY, USA, 2003.

Raphaël Clifford, Manolis Christodoulakis, Tim Crawford, David Meredith, and Geraint A Wiggins. A fast, randomised, maximal subset matching algorithm for document-level music retrieval. In *ISMIR*, pages 150–155, 2006.

Tom Collins. *Improved methods for pattern discovery in music, with applications in automated stylistic composition.* PhD thesis, The Open University, 2011.

Tom Collins and David Meredith. Maximal translational equivalence classes of musical patterns in point-set representations. In *Mathematics and Computation in Music*, pages 88–99. Springer, 2013.

Tom Collins, Jeremy Thurlow, Robin Laney, Alistair Willis, and Paul Garthwaite. A comparative evaluation of algorithms for discovering translational patterns in baroque keyboard works. 2010.

Tom Collins, Robin Laney, Alistair Willis, and Paul H. Garthwaite. Modeling pattern importance in chopin's mazurkas. *Music Perception: An Interdisciplinary Journal*, 28(4):pp. 387–414, 2011. ISSN 07307829.

Darrell Conklin. Discovery of distinctive patterns in music. *Intelligent Data Analysis*, 14(5):547–554, 2010.

Darrell Conklin and Christina Anagnostopoulou. Representation and discovery of multiple viewpoint patterns. In *Proceedings of the International Computer Music Conference*, pages 479–485. Citeseer, 2001.

John Davies, Rudi Studer, and Paul Warren. *Semantic Web technologies: trends and research in ontology-based systems.* Wiley. com, 2006.

Martin Dillon and Michael Hunter. Automated identification of melodic variants in folk music. *Computers and the Humanities*, 16(2):107–117, 1982.

S. Dixon. Onset detection revisited. In *Proceedings of the 9th International Conference on Digital Audio Effects*, pages 133–137, 2006.

J Stephen Downie. The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research. *Acoustical Science and Technology*, 29(4):247–255, 2008.

Daniel P. W. Ellis. Beat tracking by dynamic programming. *Journal of New Music Research*, 36(1):51 – 60, 2007. ISSN 09298215.

D.P.W. Ellis and G.E. Poliner. IdentifyingCover Songs' with Chroma Features and Dynamic Programming Beat Tracking. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4. IEEE, 2007.

G. Fazekas and M. Sandler. Structural decomposition of recorded vocal performances and it's application to intelligent audio editing. In *Proceedings of the 123rd AES Convention*, 2007a.

G. Fazekas and M. Sandler. Intelligent editing of studio recordings with the help of automatic music structure extraction. In *Proceedings of the 122nd AES Convention*, 2007b.

G. Fazekas and M.B. Sandler. The studio ontology framework. *ISMIR 2011 12th International Conference on Music Information Retrieval Proceedings*, 2011.

G. Fazekas, Y. Raimond, and M. Sandler. A Framework for Producing Rich Musical Metadata in Creative Music Production. In *AES 125th Convention, San Francisco*, 2008.

Gyorgy Fazekas. *Semantic audio analysis: utilities and applications.* PhD thesis, Queen Mary, University of London, 2012.

B. Fields, K. Page, D. De Roure, and T. Crawford. The segment ontology: Bridging music-generic and domain-specific. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

Derry Fitzgerald. Harmonic/percussive separation using median filtering. *Dublin Institute of Technology*, 2010.

J. Foote. Automatic audio segmentation using a measure of audio novelty. In *Proceedings of IEEE International Conference on Multimedia and Expo*, volume 1, pages 452–455, 2000.

Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C Smith. Query by humming: musical information retrieval in an audio database. In *Proceedings of the third ACM international conference on Multimedia*, pages 231–236. ACM, 1995.

M. Goto. A chorus section detection method for musical audio signals and its application to a music listening station. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1783–1794, 2006.

Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. Rwc music database: Music genre database and musical instrument sound database. In *ISMIR*, volume 3, pages 229–230, 2003.

S. Hargreaves, A. Klapuri, and M. Sandler. Structural segmentation of multitrack audio. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(10):2637 –2647, dec. 2012. ISSN 1558-7916.

Steven Hargreaves, Chris Landone, Mark Sandler, and Panos Kudumakis. Segmentation and discovery of podcast content. In *Audio Engineering Society Conference 128*. Audio Engineering Society, 2010.

Steven Hargreaves, Geraint Wiggins, and Mark Sandler. A semantic web approach to pattern discovery in data and music. In *Audio Engineering Society Conference: 53rd International Conference: Semantic Audio*. Audio Engineering Society, 2014.

Christopher Harte, Mark Sandler, Samer Abdallah, and Emilia Gómez. Symbolic representation of musical chords: A proposed syntax for text annotations. *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 66–71, 2005.

D.M. Huber and R.E. Runstein. *Modern recording techniques*. Focal press, 2005.

John Bryan Ibbotson. *A framework for the real-time analysis of musical events*. PhD thesis, University of Southampton, 2009.

Kurt Jacobson. *Connections in Music*. PhD thesis, Queen Mary, University of London, 2011.

K. Jensen, J. Xu, and M. Zachariasen. Rhythm-based segmentation of popular chinese music. In *Proc. ISMIR*. Citeseer, 2005.

A. Jordanous. Defining creativity: Finding keywords for creativity using corpus linguistics techniques. *Ventura, D.; Pease, A.; Pérez y Pérez, R*, pages 278–287, 2010.

A. Klapuri and M. Davy. *Signal processing methods for music transcription*. Springer-Verlag New York Inc, 2006.

Anssi Klapuri. *Signal processing methods for the automatic transcription of music*. Tampere University of Technology Finland, 2004.

S. Kolozali, M. Barthet, G. Fazekas, and M. Sandler. Towards the automatic generation of a semantic web ontology for musical instruments. *Semantic Multimedia*, pages 186–187, 2011.

Sefki Kolozali. *Automatic Ontology Generation Based on Semantic Audio Analysis*. PhD thesis, Queen Mary, University of London, 2013.

Kjell Lemström. *String matching techniques for music retrieval*. University of Helsinki, 2000.

Kjell Lemström and Jorma Tarhio. Transposition invariant pattern matching for multi-track strings. *Nordic Journal of Computing*, 10:185–205, 2003.

Kjell Lemström, Pauli Laine, and Sami Perttu. Using relative interval slope in music information retrieval. 1999.

F. Lerdahl and R. Jackendoff. *A generative theory of tonal music*. The MIT Press, 1996.

Bo Leuf. *The Semantic Web: crafting infrastructure for agency*. Wiley. com, 2006.

M. Levy and M. Sandler. Structural segmentation of musical audio by constrained clustering. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):318–326, 2008. ISSN 1558-7916.

M. Levy, M. Sandler, and M. Casey. Extraction of high-level musical structure from audio data and its application to thumbnail generation. In *Proc. ICASSP*, pages 1433–1436. Citeseer, 2006.

Anna Lubiw and Luke Tanur. Pattern matching in polyphonic music as a weighted geometric translation problem. In *ISMIR*, 2004.

M. Mauch, K.C. Noland, and S. Dixon. Using musical structure to enhance automatic chord transcription. In *Proc. ISMIR*, pages 231–236, 2009.

M. F. McKinney, D. Moelants, M. E. P. Davies, and A. Klapuri. Evaluation of audio beat tracking and music tempo extraction algorithms. *Journal of New Music Research*, 36(1):1 – 16, 2007. ISSN 09298215.

David Meredith, Kjell Lemström, and Geraint A Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*, 31(4):321–345, 2002. ISSN 09298215.

Marcel Mongeau and David Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175, 1990.

Eugene Narmour. *The analysis and cognition of melodic complexity: The implication-realization model*. University of Chicago Press, 1992.

Nobutaka Ono, Kenichi Miyamoto, Hirokazu Kameoka, and Shigeki Sagayama. A real-time equalizer of harmonic and percussive components in music signals. In *ISMIR*, pages 139–144, 2008.

Francois Pachet and Jean-Julien Aucouturier. Improving timbre similarity: How high is the sky? *Journal of negative results in speech and audio sciences*, 1(1):1–13, 2004.

Thomas B Passin. *Explorer's guide to the semantic web*. Manning Greenwich, 2004.

J. Paulus and A. Klapuri. Acoustic features for music piece structure analysis. In *Proc. of 11th International Conference on Digital Audio Effects*, pages 309–312. Citeseer, 2008.

J. Paulus, M. Müller, and A. Klapuri. Audio-based music structure analysis. In *Proc. 11th International Conference on Music Information Retrieval*, 2010.

G. Peeters and E. Deruty. Is Music Structure Annotation Multi-Dimensional? A Proposal for Robust Local Music Annotation. In *Proc. of 3rd Workshop on Learning the Semantics of Audio Signals*, pages 75–90. Citeseer, 2009.

G. Peeters, A. La Burthe, and X. Rodet. Toward automatic music audio summary generation from signal analysis. In *Proc. International Conference on Music Information Retrieval*, pages 94–100. Citeseer, 2002.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

E Poliner, Graham and PW Ellis, Daniel. A discriminative model for polyphonic piano transcription. *EURASIP Journal on Advances in Signal Processing*, 2007.

L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 –286, feb 1989. ISSN 0018-9219. doi: 10.1109/5.18626.

Y. Raimond, S. Abdallah, M. Sandler, and F. Giasson. The music ontology. In *Proceedings of the International Conference on Music Information Retrieval*, pages 417–422, 2007.

C. Raphael. Automatic segmentation of acoustic musical signals using hidden markov models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(4):360–370, 1999.

S. Ravuri and D.P.W. Ellis. Cover song detection: from high scores to general classification. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 65–68. IEEE, 2010.

Halfdan Rump, Shigeki Miyabe, Emiru Tsunoo, Nobutaka Ono, and Shigeki Sagayama. Autoregressive mfcc models for genre classification improved by harmonic-percussion separation. In *ISMIR*, pages 87–92, 2010.

Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of sparql query optimization. In *Proceedings of the 13th International Conference on Database Theory*, pages 4–33. ACM, 2010.

Toby Segaran, Colin Evans, and Jamie Taylor. *Programming the semantic web*. O'Reilly Media, 2009.

X Serra, M Magas, E Benetos, M Chudy, S Dixon, A Flexer, E Gómez, F Gouyon, P Herrera, S Jordà, et al. Roadmap for music information research. *Creative Commons BY-NC-ND*, 3, 2013.

Roger Shepard. Circularity in judgments of relative pitch. *The Journal of the Acoustical Society of America*, 36(12):2346–2353, 1964. doi: 10.1121/1.1919362.

David A Stech. A computer-assisted approach to micro-analysis of melodic lines. *Computers and the Humanities*, 15(4):211–221, 1981.

E. Tsunoo, T. Akase, N. Ono, and S. Sagayama. Music mood classification by rhythm and bass-line unit pattern analysis. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 265–268, 2010. doi: 10.1109/ICASSP.2010.5495964.

Rainer Typke, Panos Giannopoulos, Remco C Veltkamp, Frans Wiering, and René Van Oostrum. Using transportation distances for measuring melodic similarity. 2003.

G. Tzanetakis and P. Cook. Multifeature audio segmentation for browsing and annotation. In *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. Citeseer, 1999.

Y. Ueda, Y. Uchiyama, T. Nishimoto, N. Ono, and S. Sagayama. Hmm-based approach for automatic chord detection using refined acoustic features. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 5518–5521, 2010. doi: 10.1109/ICASSP.2010.5495218.

J. Wellhausen and M. Hoeynck. Audio thumbnailing using mpeg-7 low level audio descriptors. In *Proc. ITCom'03*. Citeseer, 2003.

G. A. Wiggins, D. Müllensiefen, and M. T. Pearce. On the non-existence of music: Why music theory is a figment of the imagination. *Musicae Scientiae*, Discussion Forum 5:231–255, 2010.

Geraint A. Wiggins. Semantic Gap?? Schemantic Schmap!! Methodological considerations in the scientific study of music. In *Proceedings of 11th IEEE International Symposium on Multimedia*, pages 477–482, 2009. ISBN 978-1-4244-5231-6. doi: 10.1109/ISM.2009.36.

T. Wilmering, G. Fazekas, and M. B. Sandler. Towards ontological representations of digital audio effects. *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11), Paris, France*, 2011.